

Troubleshooting Memory Problems

in Java Applications

Poonam Parhar
Consulting Member of Technical Staff
JVM Sustaining Engineer, Oracle

Java
Your
Next
(Cloud)

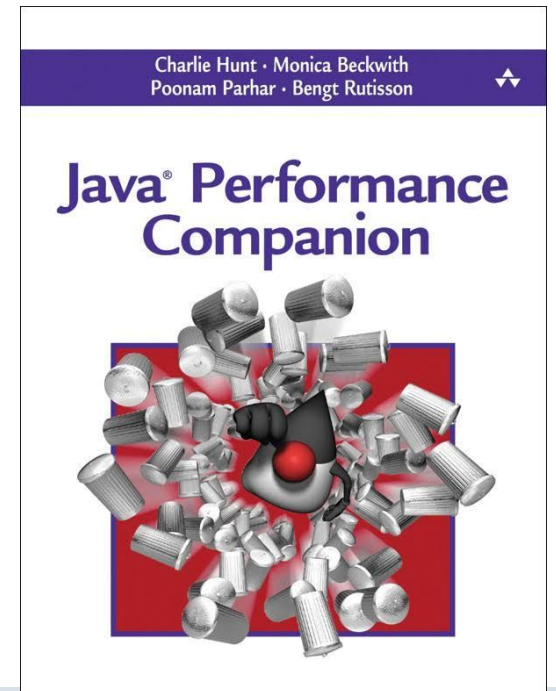


Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

About me

- Poonam Parhar
- JVM Sustaining Engineer at Oracle
- Co-author of the book 'Java Performance Companion'
- JavaOne RockStar
- Speaker at Java Conferences
- Blog: <https://blogs.oracle.com/poonam/>
- Published articles



OutOfMemoryError

Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
at java.util.Arrays.copyOfRange(Unknown Source)
at java.lang.String.<init>(Unknown Source)
at java.io.BufferedReader.readLine(Unknown Source)
at java.io.BufferedReader.readLine(Unknown Source)
at com.abc.ABCParser.dump(ABCParser.java:23)
at com.abc.ABCParser.mainABCParser.java:59)

- Application Failure with OutOfMemoryError
- Unexpected Memory Growth
- Poor Application Performance

Symptoms of

- Long GC Pauses
- Excessive GCs

Caused by

- Misconfiguration of Memory Pools
- Excessive use of Finalizers
- Explicit GC Invocations
- Memory Leak

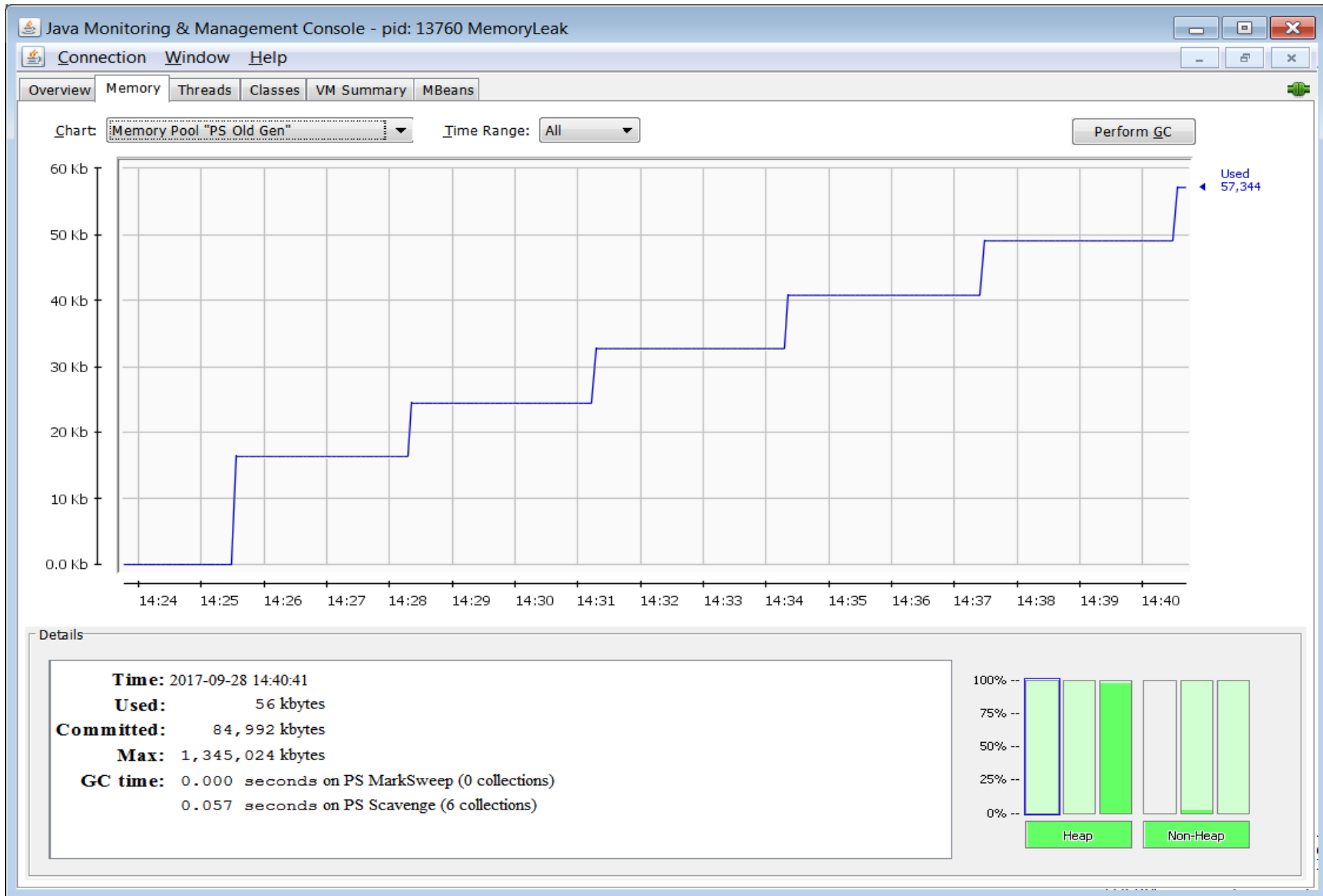
Agenda

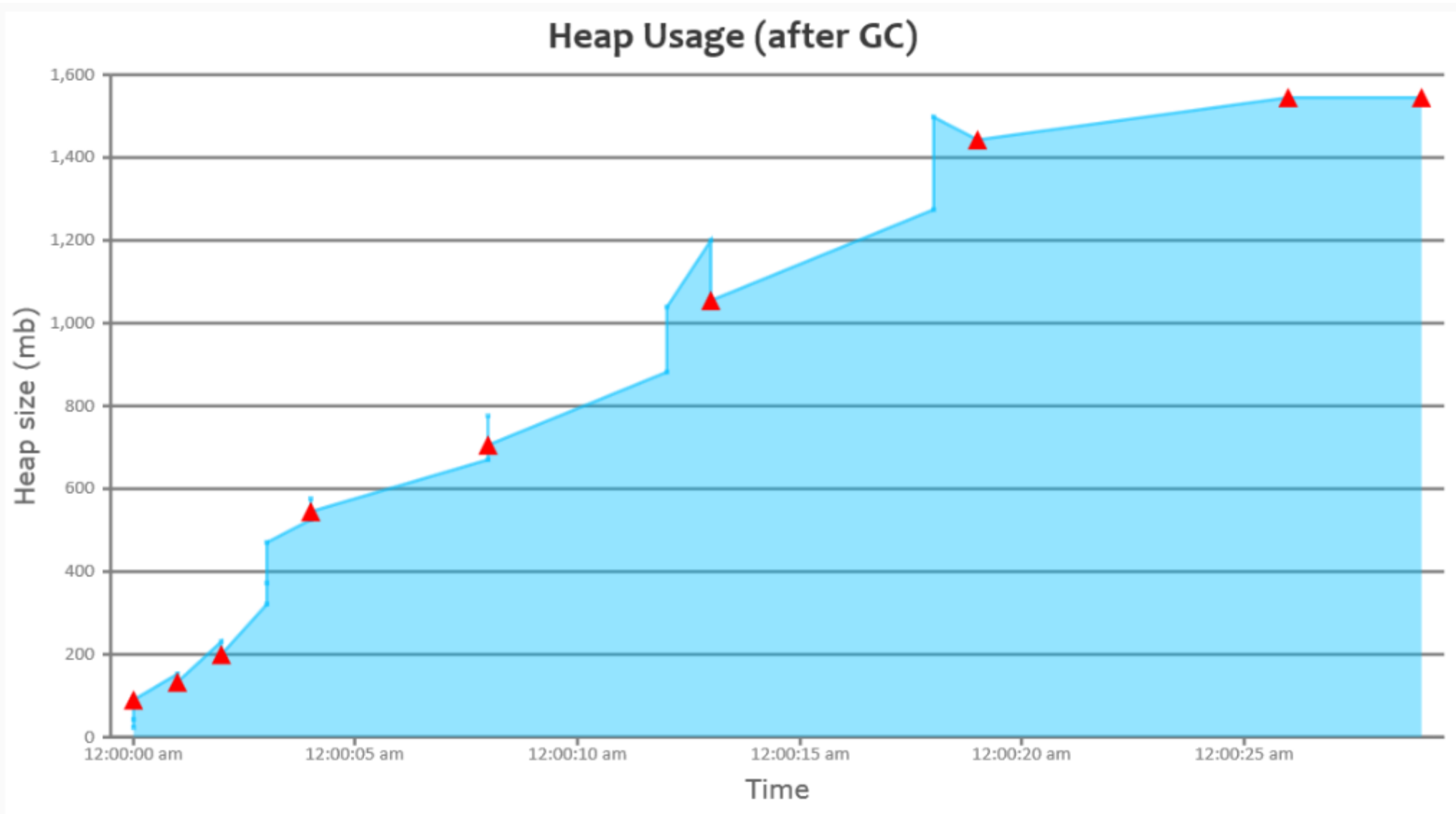
- Java Heap Memory Issues
- PermGen/Metaspace Memory Issues
- OutOfMemoryError due to Finalization
- CodeCache Issues
- Native Memory Issues

Java Heap Memory Issues

Confirm Memory Leak

- Monitor Java Heap usage over time
- If the Full GCs are not able to claim any space in the Old Gen of the heap then it could be a configuration issue
- Heap might be sized too small
- Increase the heap size and test the application again
- If there is continuous memory growth and the failure persists at the increased heap size too, there could be a memory leak





Appropriate Heap Size

`-Xmx`

Java Heap: Diagnostic Data

Java Heap: Diagnostic Data

- GC Logs
- Heap Dumps
- Heap Histograms

GC Logs

- Very helpful in determining the heap requirements
- Excessive GCs
- Long GC pauses

GC Logging Options

- Java 9:
 - G1: `-Xlog:gc*,gc+phases=debug:file=gc.log`
 - Non-G1: `-Xlog:gc*:file=gc.log`
- Prior Java Versions
 - `-XX:+PrintGCDetails`
 - `-XX:+PrintGCTimeStamps`
 - `-XX:+PrintGCDateStamps`
 - `-Xloggc:<gc log file>`

GC Logs: Heap Usage

```
2017-03-21T13:21:39.595+0000: 289634.716: [Full GC [PSYoungGen: 182784K-  
>182660K(364544K)] [ParOldGen: 1091964K->1091964K(1092096K)] 1274748K-  
>1274625K(1456640K) [PSPermGen: 493573K->493573K(494080K)], 1.2891230 secs]  
[Times: user=6.01 sys=0.00, real=1.29 secs]
```


GC Logs: Excessive GCs

```
4.381: [Full GC (Ergonomics) [PSYoungGen: 6656K->6656K(7680K)]
[ParOldGen: 16823K->16823K(17920K)] 23479K->23479K(25600K), [Metaspace: 2724K-
>2724K(1056768K)], 0.0720605 secs] [Times: user=0.23 sys=0.00, real=0.07 secs]
4.458: [Full GC (Ergonomics) [PSYoungGen: 6656K->6656K(7680K)]
[ParOldGen: 16824K->16824K(17920K)] 23480K->23480K(25600K), [Metaspace: 2724K-
>2724K(1056768K)], 0.0518873 secs] [Times: user=0.16 sys=0.00, real=0.05 secs]
4.515: [Full GC (Ergonomics) [PSYoungGen: 6656K->6656K(7680K)]
[ParOldGen: 16826K->16826K(17920K)] 23482K->23482K(25600K), [Metaspace: 2724K-
>2724K(1056768K)], 0.0530036 secs] [Times: user=0.19 sys=0.00, real=0.05 secs]
4.573: [Full GC (Ergonomics) [PSYoungGen: 6656K->6656K(7680K)]
[ParOldGen: 16827K->16827K(17920K)] 23483K->23483K(25600K), [Metaspace: 2725K-
>2725K(1056768K)], 0.0523322 secs] [Times: user=0.19 sys=0.00, real=0.05 secs]
4.631: [Full GC (Ergonomics) [PSYoungGen: 6656K->6656K(7680K)]
[ParOldGen: 16828K->16828K(17920K)] 23484K->23484K(25600K), [Metaspace: 2729K-
>2729K(1056768K)], 0.0522808 secs] [Times: user=0.17 sys=0.00, real=0.05 secs]
4.688: [Full GC (Ergonomics) [PSYoungGen: 6656K->6656K(7680K)]
[ParOldGen: 16830K->16830K(17920K)] 23486K->23486K(25600K), [Metaspace: 2729K-
>2729K(1056768K)], 0.0522224 secs] [Times: user=0.19 sys=0.00, real=0.05 secs]
4.746: [Full GC (Ergonomics) [PSYoungGen: 6656K->6656K(7680K)]
[ParOldGen: 16831K->16831K(17920K)] 23487K->23487K(25600K), [Metaspace: 2729K-
>2729K(1056768K)], 0.0528773 secs] [Times: user=0.19 sys=0.00, real=0.05 secs]
```

GC Logs: Long GC Pauses

```
2017-04-11T14:49:31.875+0000: 322836.828: [Full GC (Allocation Failure) 31G->24G(31G), 50.8369614 secs] [Eden: 0.0B(1632.0M)->0.0B(2048.0M) Survivors: 0.0B->0.0B Heap: 31.8G(31.9G)->24.5G(31.9G)], [Metaspace: 29930K->29564K(1077248K)] [Times: user=83.83 sys=0.00,real=50.84 secs]
```

Heap Dumps

- Most important diagnostic data for troubleshooting memory issues
- Can be collected using:
 - **jcmd** <pid/main class> GC.heap_dump heapdump.dmp
 - **jmap** -dump:format=b,file=snapshot.jmap <pid>
 - **JConsole** utility, using Mbean HotSpotDiagnostic
 - **Java Mission Control**, using HotSpotDiagnostic or DiagnosticCommand Mbeans
 - **-XX:+HeapDumpOnOutOfMemoryError**

Java Monitoring & Management Console - pid: 30584 MemoryLeak

Connection Window Help

Overview Memory Threads Classes VM Summary MBeans

Tree view:

- JMImplementation
 - com.sun.management
 - D diagnosticCommand
 - HotSpotDiagnostic
 - Attributes
 - Operations
 - dumpHeap
 - getVMOption
 - setVMOption
 - java.lang
 - java.nio
 - java.util.logging

Operation invocation

void p0 String , p1 true)

MBeanOperationInfo

Name	Value
Operation:	
Name	dumpHeap
Description	dumpHeap
Impact	UNKNOWN
ReturnType	void
Parameter-0:	
Name	p0
Description	p0
Type	java.lang.String
Parameter-1:	
Name	p1
Description	p1
Type	boolean

Descriptor

Name	Value
Operation:	
openType	javax.management.openmbean.SimpleType(name=java.lang.Void)
originalType	void
Parameter-0:	
openType	javax.management.openmbean.SimpleType(name=java.lang.String)
originalType	java.lang.String
Parameter-1:	
openType	javax.management.openmbean.SimpleType(name=java.lang.Boolean)
originalType	boolean

Oracle Java Mission Control

File Edit Window Help

[1.8.0_152-ea] MemoryLeak (21548)

MBean Browser

MBean Tree

Filter:

- JMImplementation
- com.sun.management
 - DiagnosticCommand
 - GarbageCollectionAggregate
 - HotSpotDiagnostic**
- java.lang
- java.nio
- java.util.logging

MBean Features

Attributes Operations Notifications Metadata

Operations

- dumpHeap : void
- getVMOption : CompositeData
- setVMOption : void

Name	Value	Description
p0	heap.dmp	p0
p1	false	p1

Execute dumpHeap("heap.dmp", false)

Overview MBean Browser Triggers System Memory Threads Diagnostic Commands

-XX:+HeapDumpOnOutOfMemoryError

```
...<several Full GCs>...
[Full GC (Ergonomics) [PSYoungGen: 12799K->12799K(14848K)] [ParOldGen: 33957K->33957K(34304K)] 46757K-
>46757K(49152K), [Metaspace: 2723K->2723K(1056768K)], 0.1026523 secs] [Times: user=0.25 sys=0.00,
real=0.09 secs]
[Full GC (Ergonomics) [PSYoungGen: 12799K->12799K(14848K)] [ParOldGen: 33958K->33946K(34304K)] 46758K-
>46746K(49152K), [Metaspace: 2723K->2723K(1056768K)], 0.1159181 secs] [Times: user=0.38 sys=0.00,
real=0.11 secs]
[Full GC (Ergonomics) [PSYoungGen: 12799K->12799K(14848K)] [ParOldGen: 33947K->33947K(34304K)] 46747K-
>46747K(49152K), [Metaspace: 2723K->2723K(1056768K)], 0.1143718 secs] [Times: user=0.36 sys=0.00,
real=0.11 secs]
[Full GC (Ergonomics) [PSYoungGen: 12799K->12799K(14848K)] [ParOldGen: 33949K->33949K(34304K)] 46749K-
>46749K(49152K), [Metaspace: 2723K->2723K(1056768K)], 0.0979753 secs] [Times: user=0.39 sys=0.00,
real=0.09 secs]
[Full GC (Ergonomics) [PSYoungGen: 12799K->12799K(14848K)] [ParOldGen: 33950K->33950K(34304K)] 46750K-
>46750K(49152K), [Metaspace: 2723K->2723K(1056768K)], 0.1008345 secs] [Times: user=0.36 sys=0.00,
real=0.10 secs]
[Full GC (Ergonomics) [PSYoungGen: 12799K->12799K(14848K)] [ParOldGen: 33951K->33951K(34304K)] 46751K-
>46751K(49152K), [Metaspace: 2723K->2723K(1056768K)], 0.0981526 secs] [Times: user=0.33 sys=0.00,
real=0.10 secs]
[Full GC (Ergonomics) [PSYoungGen: 12799K->12799K(14848K)] [ParOldGen: 33953K->33953K(34304K)] 46753K-
>46753K(49152K), [Metaspace: 2723K->2723K(1056768K)], 0.1001630 secs] [Times: user=0.39 sys=0
.00, real=0.09 secs][Full GC (Ergonomics) [PSYoungGen: 12799K->12799K(14848K)] [ParOldGen: 33954K-
>33954K(34304K)] 46754K->46754K(49152K), [Metaspace: 2723K->2723K(1056768K)], 0.0988169 secs] [Times:
user=0.39 sys=0.00, real=0.08 secs]
[Full GC (Ergonomics) [PSYoungGen: 12799K->12799K(14848K)] [ParOldGen: 33955K->33955K(34304K)] 46755K-
>46755K(49152K), [Metaspace: 2723K->2723K(1056768K)], 0.1002005 secs] [Times: user=0.31 sys=0.00,
real=0.10 secs]
[Full GC (Ergonomics) [PSYoungGen: 12799K->12799K(14848K)] [ParOldGen: 33957K->33957K(34304K)] 46757K-
>46757K(49152K), [Metaspace: 2723K->2723K(1056768K)], 0.0966616 secs] [Times: user=0.36 sys=0.00,
real=0.10 secs]
java.lang.OutOfMemoryError: GC overhead limit exceeded
Dumping heap to java_pid18904.hprof ...
```

- GC can continuously attempt to free up room on the heap by invoking frequent back-to-back Full GCs
- Even when the gains of the efforts are very little
- This impacts the performance of the application and delays the re-start of the process
- Delays the collection of heap dump

-XX:GCTimeLimit and -XX:GCHeapFreeLimit

- GCTimeLimit sets an upper limit on the amount of time that GCs can spend in percent of the total time
 - Its default value is 98%
 - Decreasing this value reduces the amount of time allowed that can be spent in the garbage collections
- GCHeapFreeLimit sets a lower limit on the amount of space that should be free after the garbage collections, represented as percent of the maximum heap
 - Its default value is 2%
 - Increasing this value means that more heap space should get reclaimed by the GCs.
- An OutOfMemoryError is thrown after a Full GC if the previous 5 consecutive GCs (could be minor or full) were not able to keep the GC cost below GCTimeLimit and were not able to free up GCHeapFreeLimit space.

Heap Histograms

- Quick glimpse of objects in Java heap
- Can be collected using:
 - `-XX:+PrintClassHistogram`, and SIGQUIT on Posix platforms and SIGBREAK on Windows
 - `jcmd` <process id/main class> GC.class_histogram filename=Myheaphistogram
 - `jmap -histo pid`
 - `jhsdb jmap` (in JDK 9)
 - **Java Mission Control** (Diagnostic Commands)

Oracle Java Mission Control

File Edit Window Help

[1.8.0_152-ea] MemoryLeak (21548)

Diagnostic Commands

Operations	Description	Value
GC.rotate_log	Inspect all objects, including unreachable objects	<Boolean>
JFR.check		
JFR.dump		
JFR.stop		
VM.check_commercial_features		
VM.command_line		
VM.flags		
VM.system_properties		
VM.unlock_commercial_features		
VM.uptime		
VM.version		
help		

Help Execute GC.class_histogram

5/5/17 9:54:30 AM Result for 'GC.class_histogram'

num	#instances	#bytes	class name
1:	870	646136	[B
2:	6935	455400	[C
3:	1891	212792	java.lang.Class
4:	6856	164544	java.lang.String
5:	1991	103192	[Ljava.lang.Object;

← Histogram

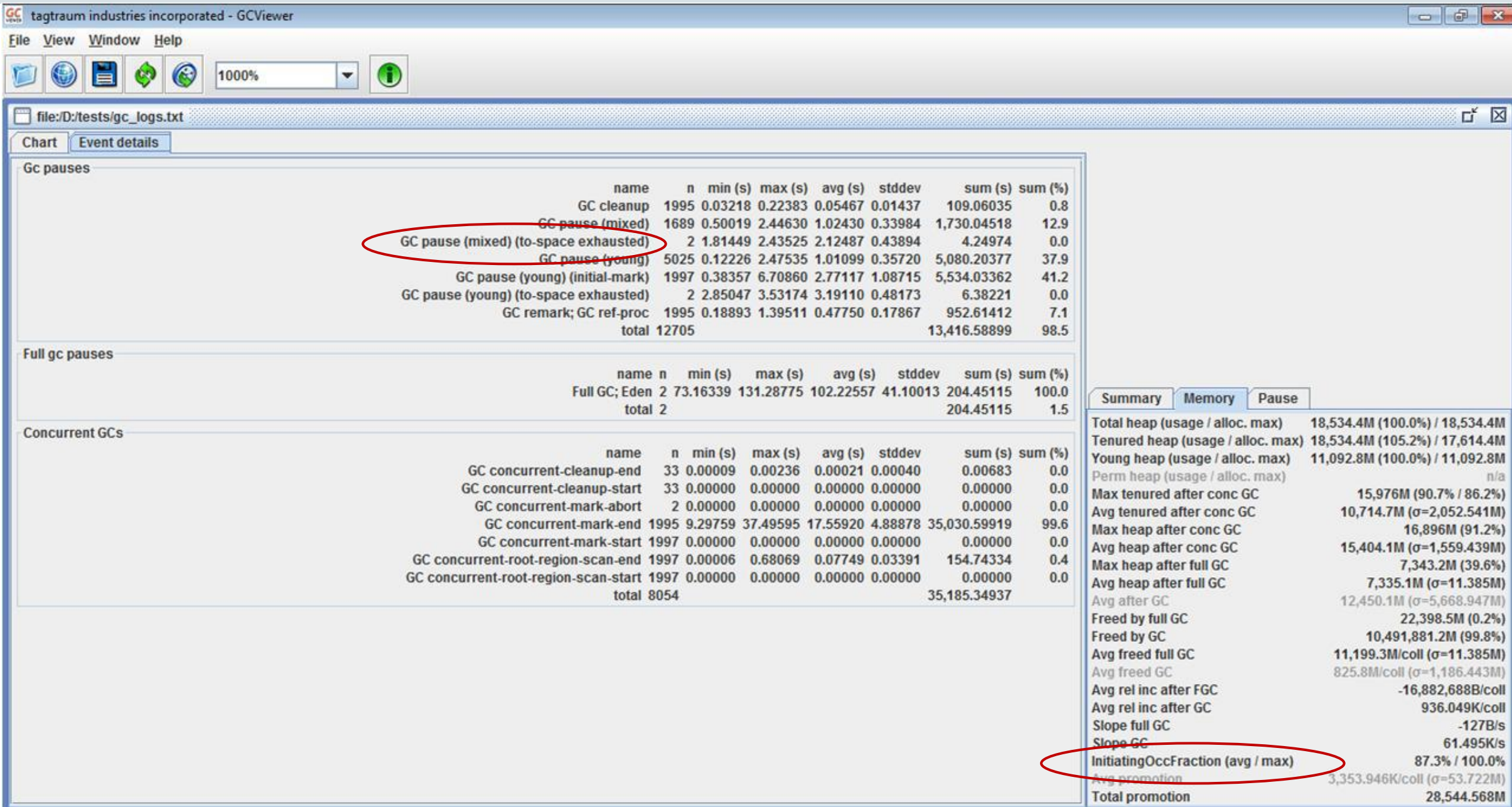
Overview MBean Browser Triggers System Memory Threads Diagnostic Commands

Java Heap: Analysis of Diagnostic Data

GC Logs Analysis

GC Logs Analysis

- What do we want to look for:
 - Are there too many Full GCs?
 - Are there GCs with long pauses?
 - Are there GCs happening too frequently?
- Manual inspection
- Automatic Analysis tools
 - Examples: GCHisto, GCViewer, gceasy.io etc.



Explicit Full GCs

164638.058: [Full GC (*System*) [PSYoungGen: 22789K->0K(992448K)] [PSOldGen: 1645508K->1666990K(2097152K)] 1668298K->1666990K(3089600K) [PSPermGen: 164914K->164914K(166720K)], **5.7499132 secs**] [Times: user=5.69 sys=0.06, real=5.75 secs]

- `-Dsun.rmi.dgc.server.gcInterval=n -Dsun.rmi.dgc.client.gcInterval=n`
- `-XX:+PrintClassHistogram` and SIGBREAK /SIGQUIT signal
- **Solution:** `-XX:+DisableExplicitGC`

Heap Dump Analysis

Eclipse MAT - Memory Analyzer Tool

- Community developed tool for analyzing heap dumps
- Some of the amazing features that it offers are:
 - Leak Suspects
 - Histograms
 - Unreachable objects
 - Duplicate Classes
 - Path to GC roots
 - OQL

Eclipse Memory Analyzer

File Edit Window Help

Inspector

@ 0xfe686a08

byte[]

class java.lang.Class @ 0xfe696620

java.lang.Object

java.lang.ClassLoader @ 0x0

0 (shallow size)

0 (retained size)

no GC root

Statics Attributes »2

Type	Name	Value
------	------	-------

cbfe1M1N5_heap.hprof jmap.cbfe2.10461.heap.hprof.2017-03-13_19-51-58.hprof help

Overview default_report org.eclipse.... Histogram list_objects [selection of 'b... merge_shortest_paths [selecti...

Class Name	Objects	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>	<Numeric>
byte[]	664	406,304	>= 406,304
char[]	4,366	309,784	>= 309,784
java.lang.String	4,332	103,968	>= 369,248
java.lang.reflect.Method	458	40,304	>= 87,144
java.lang.Object[]	911	39,936	>= 310,160
java.util.HashMap\$Node	1,147	36,704	>= 201,800
int[]	450	35,160	>= 35,160
java.util.HashMap\$Node[]	198	20,360	>= 222,696
java.util.TreeMap\$Entry	446	17,840	>= 33,840
java.lang.Class	1,758	16,464	>= 590,472
java.util.HashMap	323	15,504	>= 227,656
java.util.concurrent.ConcurrentHashMap\$Node	435	13,920	>= 40,368
java.lang.String[]	391	13,576	>= 54,176
java.lang.ref.SoftReference	288	11,520	>= 12,456
sun.misc.FDBigInteger	341	10,912	>= 37,424
java.lang.Long	441	10,584	>= 10,624
java.util.LinkedHashMap\$Entry	256	10,240	>= 42,848
java.util.Hashtable\$Entry	300	9,600	>= 23,920
java.lang.Class[]	368	9,488	>= 9,488
java.util.LinkedList\$Node	392	9,408	>= 9,408
java.util.Hashtable\$Entry[]	76	6,440	>= 30,360
java.lang.invoke.LambdaForm\$Name	198	6,336	>= 12,624
java.util.LinkedList	198	6,336	>= 15,752
java.lang.invoke.MethodType\$ConcurrentWeakInternSet\$WeakEn...	186	5,952	>= 7,552
java.lang.invoke.MemberName	186	5,952	>= 12,112
java.lang.ref.SoftReference[]	94	5,640	>= 9,728
java.util.WeakHashMap\$Entry	141	5,640	>= 11,400
java.lang.invoke.MethodType	136	5,440	>= 42,360

674M of 1171M

Eclipse Memory Analyzer

File Edit Window Help

Inspector

@ 0xfe7bd378
 byte[]
 class byte[] @ 0xfe686a08
 java.lang.Object
 java.lang.ClassLoader @ 0x0
 1,040 (shallow size)
 1,040 (retained size)
 no GC root

Statics Attributes »
 Type Name Value
 byte [0] 0
 byte [1] 0
 byte [2] 0
 byte [3] 0
 byte [4] 0
 byte [5] 0
 byte [6] 0
 byte [7] 0
 byte [8] 0
 byte [9] 0
 byte [10] 0
 byte [11] 0
 byte [12] 0
 byte [13] 0
 byte [14] 0
 byte [15] 0
 byte [16] 0
 byte [17] 0
 byte [18] 0
 byte [19] 0

cbfe1M1N5_heap.hprof jmap.cbfe2.10461.heap.hprof.2017-03-13_19-51-58.hprof help

Overview default_report org.... Histogram list_objects [select...] merge_shortest_paths... list_objects [select...]

Class Name	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>
byte[16] @ 0xfe7c4040)%....g	32	32
byte[16] @ 0xfe7c3fc0:7.M	32	32
byte[16] @ 0xfe7c3f40 &...A...p....:7.M	32	32
byte[16] @ 0xfe7c3ec0 &...A...p..E..v	32	32
byte[8] @ 0xfe7c2a38	24	24
byte[80] @ 0xfe7c2928	96	96
byte[8] @ 0xfe7c2910	24	24
byte[80] @ 0xfe7c2800 192.168.56.1.....	96	96
byte[8192] @ 0xfe7c07f0 Q....w...':=[..> ..nsr..javax.management.AttributeList.kH..H'I...	8,208	8,208
byte[8] @ 0xfe7c07d8	24	24
byte[26] @ 0xfe7c07a8 ..192.168.56.1.....	48	48
byte[8192] @ 0xfe7be608 P....w"3(...G.r.'=[..>W9.j.#\Gsr..javax.management.Object	8,208	8,208
byte[1] @ 0xfe7be5a8 .	24	24
byte[1024] @ 0xfe7bdff0	1,040	1,040
byte[1024] @ 0xfe7bdbcb8	1,040	1,040
byte[1024] @ 0xfe7bd7a0	1,040	1,040
byte[1024] @ 0xfe7bd378	1,040	1,040
byte[1024] @ 0xfe7bcf50	1,040	1,040
byte[1024] @ 0xfe7bcb28	1,040	1,040
byte[1024] @ 0xfe7bc700	1,040	1,040
byte[1024] @ 0xfe7bc2d8	1,040	1,040
byte[1024] @ 0xfe7bbeb0	1,040	1,040
byte[1024] @ 0xfe7bba88	1,040	1,040
byte[1024] @ 0xfe7bb660	1,040	1,040
byte[1024] @ 0xfe7bb238	1,040	1,040
byte[1024] @ 0xfe7bae10	1,040	1,040
byte[1024] @ 0xfe7ba9e8	1,040	1,040
byte[1024] @ 0xfe7ba5c0	1,040	1,040
byte[1024] @ 0xfe7ba198	1,040	1,040
byte[1024] @ 0xfe7b9d70	1,040	1,040

738M of 1171M

Eclipse Memory Analyzer

File Edit Window Help

Inspector

@ 0xfe7bd378

byte[]

class byte[] @ 0xfe686a08

java.lang.Object

java.lang.ClassLoader @ 0x0

1,040 (shallow size)

1,040 (retained size)

no GC root

Statics Attributes »2

Type	Name	Value
byte	[0]	0
byte	[1]	0
byte	[2]	0
byte	[3]	0
byte	[4]	0
byte	[5]	0
byte	[6]	0
byte	[7]	0
byte	[8]	0
byte	[9]	0
byte	[10]	0
byte	[11]	0
byte	[12]	0
byte	[13]	0
byte	[14]	0
byte	[15]	0
byte	[16]	0
byte	[17]	0
byte	[18]	0
byte	[19]	0

cbfe1M1N5_heap.hprof jmap.cbfe2.10461.heap.hprof.2017-03-13_19-51-58.hprof help

Overview default_report org.... Histogram list_objects [select...] merge_shortest_paths... list_objects [select...]

Class Name	Ref. Objects	Shallo...	Ref. Sha...	Retained Heap
<Regex>	<Numeric>	<Num...	<Numeri...	<Numeric>
java.lang.Thread @ 0xfe6eb268 main Thread	1	120	1,040	228,464
<Java Local> MemoryLeak @ 0xfe717748	1	16	1,040	227,976
longLivedObjects java.util.Vector @ 0xfe717b10	1	32	1,040	227,960
elementData java.lang.Object[320] @ 0xfe7b02a8	1	1,296	1,040	227,928
[163] MemoryLeak\$HeapObject @ 0xfe7bd360	1	24	1,040	1,064
b byte[1024] @ 0xfe7bd378	1	1,040	1,040	1,040

735M of 1171M

PermGen/MetaSpace Memory Issues

Confirm Memory Leak

- Monitor PermGen/Metaspace usage over time
- If the Full GCs are not able to claim any space in the PermGen/Metaspace then it could be a configuration issue
- PermGen/Metaspace might be sized too small
- Increase the PermGen/Metaspace size and test the application again
- If there is continuous memory growth and the failure persists at the increased PermGen/Metaspace size too, there could be a memory leak

Configure PermGen Size

`-XX:PermSize=m -XX:MaxPermSize=n`

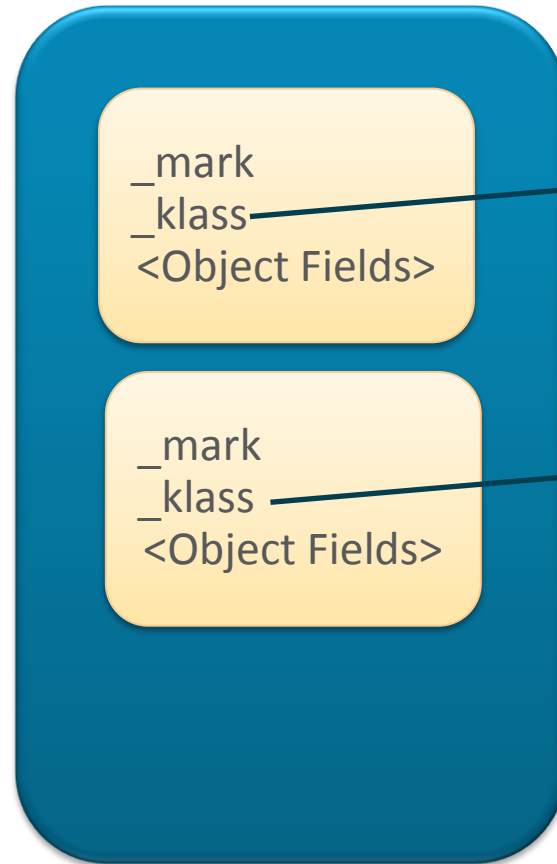
Configure Metaspace Size

`-XX:MetaspaceSize=m -XX:MaxMetaspaceSize=n`

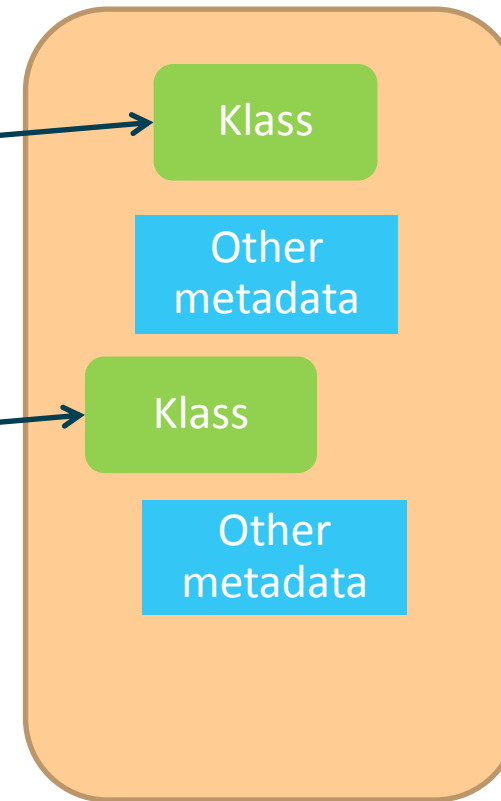
OutOfMemoryError: Compressed class space

- If UseCompressedClassPointers is enabled, then two separate areas of native memory are used for class metadata
 - By default UseCompressedClassPointers is ON if UseCompressedOops is ON
- 64-bit class pointers are represented by 32-bit offsets
- 32-bit offsets can be used to reference class-metadata stored in the 'compressed class space'
- By default, 1GB of address space is reserved for the compressed class space. This can be configured using **CompressedClassSpaceSize**.
- MaxMetaspaceSize sets an upper limit on the total committed size of both of these regions
 - committed size of compressed class space + committed size of Metaspace <= MaxMetaspaceSize

Java Heap

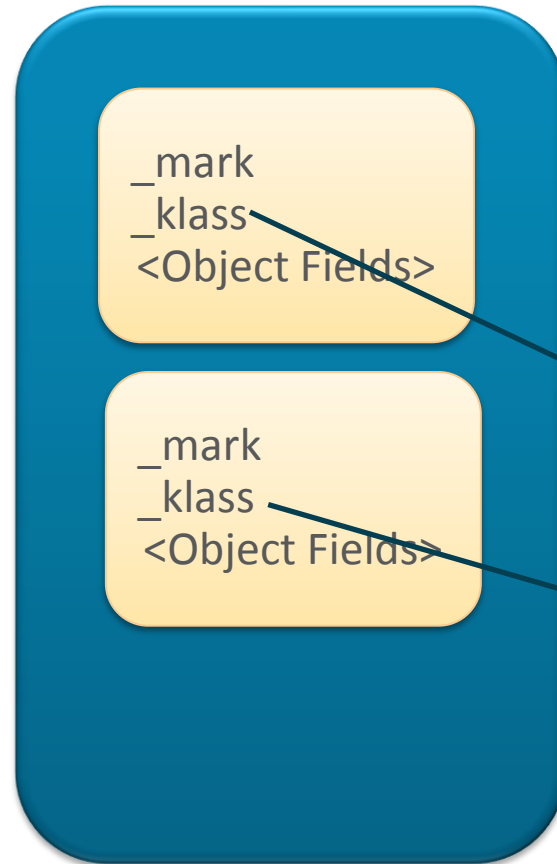


Metaspace

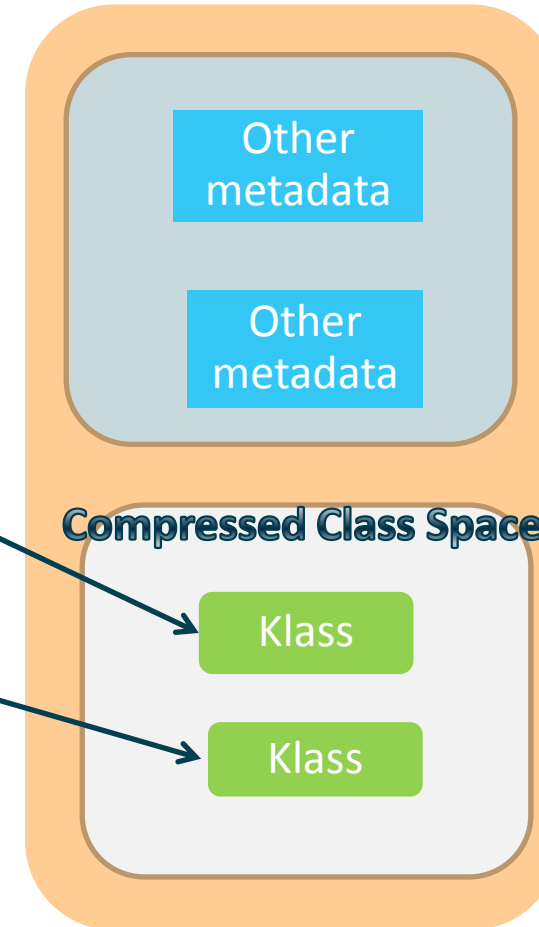


+UseCompressedClassPointers

Java Heap



Metaspace



GC log with +UseCompressedClassPointers

Metaspace used 2921K, capacity 4486K, committed 4864K, reserved 1056768K
class space used 288K, capacity 386K, committed 512K, reserved 1048576K

PermGen/MetaSpace: Diagnostic Data collection and Analysis

PermGen/MetaSpace: Diagnostic Data collection and Analysis

- GC logs including options:
 - `-verbose:class`or
 - `-XX:+TraceClassLoading -XX:+TraceClassUnloading`
- Data collected with:
 - `jmap -permstat` (up to JDK 7)
 - `jmap -clstats` (JDK 8 onwards)
- Heap Dumps
- JDK 8: class statistics information with `'jcmd <pid> GC.class_stats'`

Make sure that classes get unloaded

- Ensure `-Xnoclassgc` is not in use
- Ensure that `-XX:+CMSClassUnloadingEnabled` is used when using CMS on Java 6 or 7

-verbose:class

```
[Loading weblogic.i18n.logging.MessageDispatcher from
  file:/opt/weblogic1213/wlserver/modules/features/weblogic.server.merged.jar]
[Loaded weblogic.i18n.logging.MessageDispatcher from
  file:/opt/weblogic1213/wlserver/modules/features/weblogic.server.merged.jar]
[Loaded weblogic.i18n.logging.CoreEnginePrimordialLoggerWrapper from
  file:/opt/weblogic1213/wlserver/modules/features/weblogic.server.merged.jar]
[Loading weblogic.logging.WLMessageLogger from
  file:/opt/weblogic1213/wlserver/modules/features/weblogic.server.merged.jar]
...
[Loading sun.reflect.GeneratedMethodAccessor486 from __JVM_DefineClass__]
[Loaded sun.reflect.GeneratedMethodAccessor486 from __JVM_DefineClass__]
[Loading sun.reflect.GeneratedMethodAccessor487 from __JVM_DefineClass__]
[Loaded sun.reflect.GeneratedMethodAccessor487 from __JVM_DefineClass__]
[Loading sun.reflect.GeneratedMethodAccessor488 from __JVM_DefineClass__]
[Loaded sun.reflect.GeneratedMethodAccessor488 from __JVM_DefineClass__]
[Loading sun.reflect.GeneratedMethodAccessor489 from __JVM_DefineClass__]
[Loaded sun.reflect.GeneratedMethodAccessor489 from __JVM_DefineClass__]
...
[Unloading class sun.reflect.GeneratedMethodAccessor489 0x000000010128fc30]
[Unloading class sun.reflect.GeneratedMethodAccessor488 0x000000010128f830]
[Unloading class sun.reflect.GeneratedMethodAccessor487 0x000000010128f430]
[Unloading class sun.reflect.GeneratedMethodAccessor486 0x000000010128f030]
[Unloading class sun.reflect.GeneratedMethodAccessor482 0x000000010128e030]
[Unloading class sun.reflect.GeneratedMethodAccessor481 0x000000010128dc30]
[Unloading class sun.reflect.GeneratedSerializationConstructorAccessor297 0x0000000101274c30]
[Unloading class sun.reflect.GeneratedSerializationConstructorAccessor296 0x0000000101274830]
```

GC Logs

```
74062.764: [Full GC (Last ditch collection) 74062.764: [CMS: 1444990K-
>1444951K(2599396K), 6.0356492 secs] 1444990K->1444951K(3046692K), [Metaspace:
4050878K->4050878K(6356992K)], 6.0366164 secs] [Times: user=6.04 sys=0.01, real=6.04
secs]

74068.804: [GC (CMS Initial Mark) [1 CMS-initial-mark: 1444951K(2599396K)]
1445248K(3046692K), 0.9821073 secs] [Times: user=1.23 sys=0.00, real=0.98 secs]
74069.787: [CMS-concurrent-mark-start]

74069.818: [Full GC (Metadata GC Threshold) 74069.818: [CMS74071.433: [CMS-concurrent-
mark: 1.642/1.647 secs] [Times: user=3.33 sys=0.00, real=1.65 secs] (concurrent mode
failure): 1444951K->1444879K(2599396K), 7.5785637 secs] 1445513K->1444879K(3046692K),
[Metaspace: 4050878K->4050878K(6356992K)], 7.5795618 secs] [Times: user=9.19 sys=0.00,
real=7.58 secs]

java.lang.OutOfMemoryError: Compressed class space
```


jmap -permstat

```
$ jmap -permstat 29620
Attaching to process ID 29620, please wait...
Debugger attached successfully. Client compiler detected.
JVM version is 24.85-b06
12674 intern Strings occupying 1082616 bytes. finding class loader instances ..
done. computing per loader stat ..done. please wait.. computing liveness.....done.
class_loader      classes bytes parent_loader  alive?  type
<bootstrap> 1846 5321080 null live <Internal>
0xd0bf3828 0 0 null live sun/misc/Launcher$ExtClassLoader@0xd8c98c78
0xd0d2f370 1 904 null dead sun/reflect/DelegatingClassLoader@0xd8c22f50
0xd0c99280 1 1440 null dead sun/reflect/DelegatingClassLoader@0xd8c22f50
0xd0b71d90 0 0 0xd0b5b9c0 live java/util/ResourceBundle$RBCClassLoader@0xd8d042e8
0xd0d2f4c0 1 904 null dead sun/reflect/DelegatingClassLoader@0xd8c22f50
0xd0b5bf98 1 920 0xd0b5bf38 dead sun/reflect/DelegatingClassLoader@0xd8c22f50
0xd0c99248 1 904 null dead sun/reflect/DelegatingClassLoader@0xd8c22f50
0xd0d2f488 1 904 null dead sun/reflect/DelegatingClassLoader@0xd8c22f50
0xd0b5bf38 6 11832 0xd0b5b9c0 dead sun/reflect/misc/MethodUtil@0xd8e8e560
0xd0d2f338 1 904 null dead sun/reflect/DelegatingClassLoader@0xd8c22f50
0xd0d2f418 1 904 null dead sun/reflect/DelegatingClassLoader@0xd8c22f50
0xd0d2f3a8 1 904 null dead sun/reflect/DelegatingClassLoader@0xd8c22f50
0xd0b5b9c0 317 1397448 0xd0bf3828 live sun/misc/Launcher$AppClassLoader@0xd8cb83d8
0xd0d2f300 1 904 null dead sun/reflect/DelegatingClassLoader@0xd8c22f50
0xd0d2f3e0 1 904 null dead sun/reflect/DelegatingClassLoader@0xd8c22f50
0xd0ec3968 1 1440 null dead sun/reflect/DelegatingClassLoader@0xd8c22f50
0xd0e0a248 1 904 null dead sun/reflect/DelegatingClassLoader@0xd8c22f50
0xd0c99210 1 904 null dead sun/reflect/DelegatingClassLoader@0xd8c22f50
0xd0d2f450 1 904 null dead sun/reflect/DelegatingClassLoader@0xd8c22f50
0xd0d2f4f8 1 904 null dead sun/reflect/DelegatingClassLoader@0xd8c22f50
0xd0e0a280 1 904 null dead sun/reflect/DelegatingClassLoader@0xd8c22f50

total = 22          2186      6746816      N/A      alive=4, dead=18          N/A
```

jmap -clstats

```
jmap -clstats 26240
```

```
Attaching to process ID 26240, please wait...
```

```
Debugger attached successfully. Server compiler detected. JVM version is 25.66-b00
```

```
finding class loader instances ..done. computing per loader stat ..done. please wait..
```

```
computing liveness.liveness analysis may be inaccurate ...
```

class_loader	classes	bytes	parent_loader	alive?	type
<bootstrap>	513	950353	null	live	<internal>
0x0000000084e066d0	8	24416	0x0000000084e06740	live	
sun/misc/Launcher\$AppClassLoader@0x0000000016bef6a0					
0x0000000084e06740	0	0	null	live	
sun/misc/Launcher\$ExtClassLoader@0x0000000016befa48					
0x0000000084ea18f0	0	0	0x0000000084e066d0	dead	
java/util/ResourceBundle\$RBCClassLoader@0x0000000016c33930					

Heap Dumps

- Heap Dumps help here too
- Look for classes that should have been unloaded
- Eclipse MAT offers a very nice feature called ‘Duplicate Classes’
 - Displays classes that were loaded multiple times by different classloader instances
 - If duplicate classes keep growing over time/red deployments, it’s a red flag

Eclipse Memory Analyzer

File Edit Window Help

Inspector

@ 0xffffffff842495a8

- JaxbClassLoader
- org.eclipse.persistence.internal.jaxb
- class org.eclipse.persistence.internal.jaxb...
- java.lang.ClassLoader
- sun.misc.Launcher\$AppClassLoader @ 0...

72 (shallow size)

10,320 (retained size)

no GC root

Attributes Class Hierarchy Value »1

Type	Name	Value
ref	generatedClassC...	18
ref	generatedClasses	java.util.HashM
ref	classAssertionSt...	null
ref	packageAssertio...	null
boole...	defaultAssertion...	false
ref	assertionLock	org.eclipse.pers
ref	nativeLibraries	java.util.Vector
ref	packages	java.util.HashM
ref	domains	java.util.HashSe
ref	defaultDomain	java.security.Pro
ref	classes	java.util.Vector
ref	package2certs	java.util.Hashtal
ref	parallelLockMap	null
ref	parent	weblogic.utils.c

cbfe1M1N5_heap.hprof

Overview duplicate_classes

Class Name	Count	Define...	No. o...
<Regex>	<Numeric>	<Num...	<Num...
java.lang.invoke.LambdaForm\$DMH	34		
java.lang.invoke.LambdaForm\$SMH	29		
org.eclipse.persistence.jaxb.generated0	9		
org.eclipse.persistence.internal.jaxb.JaxbClassLoader @ 0xffffffff842495a8		18	0
org.eclipse.persistence.internal.jaxb.JaxbClassLoader @ 0xffffffff842c5128		31	0
org.eclipse.persistence.internal.jaxb.JaxbClassLoader @ 0xffffffff8439bd08		3	0
org.eclipse.persistence.internal.jaxb.JaxbClassLoader @ 0xffffffff844dc550		4	0
org.eclipse.persistence.internal.jaxb.JaxbClassLoader @ 0xffffffff86134b08		2	0
org.eclipse.persistence.internal.jaxb.JaxbClassLoader @ 0xffffffff8619caa8		2	0
org.eclipse.persistence.internal.jaxb.JaxbClassLoader @ 0xffffffff8622a6d8		2	0
org.eclipse.persistence.internal.jaxb.JaxbClassLoader @ 0xffffffff86357748		2	0
org.eclipse.persistence.internal.jaxb.JaxbClassLoader @ 0xffffffff8636e670		2	0
Total: 9 entries			
org.eclipse.persistence.jaxb.generated1	9		
org.eclipse.persistence.internal.jaxb.JaxbClassLoader @ 0xffffffff842495a8		18	0
org.eclipse.persistence.internal.jaxb.JaxbClassLoader @ 0xffffffff842c5128		31	0
org.eclipse.persistence.internal.jaxb.JaxbClassLoader @ 0xffffffff8439bd08		3	0
org.eclipse.persistence.internal.jaxb.JaxbClassLoader @ 0xffffffff844dc550		4	0
org.eclipse.persistence.internal.jaxb.JaxbClassLoader @ 0xffffffff86134b08		2	0
org.eclipse.persistence.internal.jaxb.JaxbClassLoader @ 0xffffffff8619caa8		2	0
org.eclipse.persistence.internal.jaxb.JaxbClassLoader @ 0xffffffff8622a6d8		2	0
org.eclipse.persistence.internal.jaxb.JaxbClassLoader @ 0xffffffff86357748		2	0
org.eclipse.persistence.internal.jaxb.JaxbClassLoader @ 0xffffffff8636e670		2	0
Total: 9 entries			
java.lang.invoke.LambdaForm\$BMH	5		
org.eclipse.persistence.jaxb.generated2	4		
org.eclipse.persistence.jaxb.generated3	3		
org.apache.commons.logging.Log	2		
org.apache.commons.logging.LogConfigurationException	2		
org.apache.commons.logging.LogFactory	2		
org.apache.commons.logging.LogFactory\$1	2		
org.apache.commons.logging.LogFactory\$2	2		
org.apache.commons.logging.LogFactory\$3	2		
org.apache.commons.logging.LogFactory\$4	2		

882M of 1240M

OutOfMemoryError due to Finalization

Finalization

- OutOfMemoryError can also be caused due to excessive use of finalizers
- Objects with a **finalizer** (i.e. a **finalize()** method) may delay the reclamation of the space occupied by them
- Finalizer thread needs to invoke the finalize() method of the instances before those instances can be reclaimed
- If the Finalizer thread does not keep up with the rate at which the objects become eligible for finalization, JVM might fail with an OutOfMemoryError
- Deprecated in Java 9
- <https://bugs.openjdk.java.net/browse/JDK-8165641>

Finalization: Diagnostic Data and Tools

Finalization: Diagnostic Data and Tools

- JConsole
- jmap -finalizerinfo
- Heap Dumps

Java Monitoring & Management Console - pid: 23336 MemoryLeak

Connection Window Help

Overview Memory Threads Classes VM Summary MBeans

VM Summary

Monday, May 8, 2017 9:58:59 AM PDT

Connection name: pid: 23336 MemoryLeak **Uptime:** 14 minutes

Virtual Machine: Java HotSpot(TM) 64-Bit Server VM version 25.152-b01 **Process CPU time:** 6.318 seconds

Vendor: Oracle Corporation **JIT compiler:** HotSpot 64-Bit Tiered Compilers

Name: 23336@POBAJAJ-LAP **Total compile time:** 3.190 seconds

Live threads: 15 **Current classes loaded:** 2,593

Peak: 15 **Total classes loaded:** 2,716

Daemon threads: 14 **Total classes unloaded:** 123

Total threads started: 21

Current heap size: 11,393 kbytes **Committed memory:** 25,088 kbytes

Maximum heap size: 25,088 kbytes **Pending finalization:** 0 objects

Garbage collector: Name = 'PS MarkSweep', Collections = 2, Total time spent = 0.062 seconds

Garbage collector: Name = 'PS Scavenge', Collections = 3, Total time spent = 0.028 seconds

Operating System: Windows 7 6.1 **Total physical memory:** 8,068,736 kbytes

Architecture: amd64 **Free physical memory:** 2,623,304 kbytes

Number of processors: 4 **Total swap space:** 16,135,612 kbytes

Committed virtual memory: 102,012 kbytes **Free swap space:** 7,954,368 kbytes

VM arguments: -Xmx25m -XX:NewSize=15m

Class path: .

Library path: d:\Java\jdk1.8.0_152\bin;C:\windows\Sun\Java\bin;C:\windows\system32;C:\windows;d:\Java\jdk1.8.0_152\bin;C:\ProgramData\Oracle\Java\javapath;C:\windows\system32;C:\windows;C:\windows\System32\Wbem;C:\windows\System32\WindowsPowerShell\v1.0\;C:\Program Files (x86)\Windows Kits\8.1\Windows Performance Toolkit\;C:\Program Files\Microsoft SQL Server\110\Tools\Binn\;C:\Program Files\TortoiseHg\.

Boot class path: d:\Java\jdk1.8.0_152\jre\lib\resources.jar;d:\Java\jdk1.8.0_152\jre\lib\rt.jar;d:\Java\jdk1.8.0_152\jre\lib\sunrsasign.jar;d:\Java\jdk1.8.0_152\jre\lib\jsse.jar;d:\Java\jdk1.8.0_152\jre\lib\jce.jar;d:\Java\jdk1.8.0_152\jre\lib\charsets.jar;d:\Java\jdk1.8.0_152\jre\lib\jfr.jar;d:\Java\jdk1.8.0_152\jre\classes

Eclipse Memory Analyzer

File Edit Window Help

Inspector

@ 0xff0c7000

- Finalizer\$FinalizerThread
- java.lang.ref
- class java.lang.ref.Finalizer\$FinalizerTh...
- java.lang.Thread
- java.lang.ClassLoader @ 0x0
- 128 (shallow size)
- 248 (retained size)
- GC root: Thread

Statics Attributes >>2

Type	Name	Value
boolean	running	true
int	threadLocalRand...	0
int	threadLocalRand...	0
long	threadLocalRand...	0
ref	uncaughtExcepti...	null
ref	blockerLock	java.lang.C...
ref	blocker	null
ref	parkBlocker	null
int	threadStatus	401
long	tid	3
long	nativeParkEvent...	0
long	stackSize	0
ref	inheritableThrea...	null
ref	threadLocals	null
ref	inheritedAccess...	java.securi...
ref	contextClassLoa...	null
ref	group	system
ref	target	null
boolean	stillborn	false
boolean	daemon	true
boolean	single_step	false
long	eetop	139339776
ref	threadQ	null
int	priority	8

java_pid18516.hprof

Overview finalizer_overview finalizer_in_processing

Finalizers

▼ In processing by Finalizer Thread

Class Name	Shallow Heap	Retained Heap
java.lang.System\$2 @ 0xff003658 >	16	16

▼ Ready for Finalizer Thread - Object List

Class Name	Shallow Heap	Retained Heap
------------	--------------	---------------

▼ Ready for Finalizer Thread - Histogram

Class Name	Objects	Shallow Heap	Retained Heap
------------	---------	--------------	---------------

▼ Finalizer Thread

Object / Stack Frame	Name	Shallow Heap	Retained Heap	Context Class Loader	Is Daemon	Details
java.lang.ref.Finalizer\$FinalizerThread @ 0xff0c7000 >	Finalizer	128	248		true	Thread Details

▼ Finalizer Thread Locals

Class Name	Shallow Heap	Retained Heap
------------	--------------	---------------

713M of 1194M

CodeCache is full. Compiler has been disabled

CodeCache is full. Compiler has been disabled

- CodeCache is the memory pool to store the compiled code generated by the JIT compilers
- There is no specific OutOfMemoryError thrown when CodeCache is full
- Managed by Sweeper
- An emergency CodeCache cleanup is done when it becomes full
 - This may discard the compiled code that might be needed shortly after
 - Compiler needs to work again to generate the compiled code for those methods
- Ensure CodeCache size is sufficient
 - Increase CodeCache maximum size with **ReservedCodeCacheSize**

OutOfMemoryError: Native Memory

Native OutOfMemoryError

```
# A fatal error has been detected by the Java  
Runtime Environment:  
#  
# java.lang.OutOfMemoryError : unable to create  
new native Thread
```

```
# A fatal error has been detected by the Java  
Runtime Environment:  
#  
# java.lang.OutOfMemoryError: requested 32756  
bytes for ChunkPool::allocate. Out of swap  
space?  
#  
# Internal Error (allocation.cpp:166),  
pid=2290, tid=27 # Error: ChunkPool::allocate
```

Native OutOfMemoryError

- JVM is not able to allocate from native memory
 - Not managed by the JVM
- This process or other processes on the system are eating up the native memory
- Can make more room for native heap by:
 - Reducing the Java Heap, PermGen/Metaspace, number of threads and/or their stack sizes etc.
 - Reducing the number of processes running on the system
- If the above don't help, we might be facing a native heap memory leak
 - Example: JNI code allocating native buffers

Native Memory: Diagnostic Data

Native Memory: Diagnostic Data

- Native memory leaks in the JVM
 - Native Memory Tracker output
- Native Memory Leaks outside JVM
 - Process map output with tools like **pmap**
 - Results from native memory leak tools **such as libumem, valgrind** etc.
 - Core file

Native Memory Tracker

- The Native Memory Tracker (NMT) can be used to track native memory that is **used internally by the JVM**
- It cannot track memory allocated outside the JVM or by native libraries

NMT

- Start the process with NMT enabled using **NativeMemoryTracking**
- The output level can be set to a 'summary' or 'detail' level:
 - -XX:NativeMemoryTracking=summary
 - -XX:NativeMemoryTracking=detail
- Use jcmd to get the native memory usage details:
 - jcmd <pid> VM.native_memory

```
jcmd 90172 VM.native_memory 90172:
```

Native Memory Tracking:

Total: reserved=3431296KB, committed=2132244KB

- Java Heap (reserved=2017280KB, committed=2017280KB)
 (mmap: reserved=2017280KB, committed=2017280KB)
- Class (reserved=1062088KB, committed=10184KB)
 (classes #411)
 (malloc=5320KB #190)
 (mmap: reserved=1056768KB, committed=4864KB)
- Thread (reserved=15423KB, committed=15423KB)
 (thread #16)
 (stack: reserved=15360KB, committed=15360KB)
 (malloc=45KB #81)
 (arena=18KB #30)
- Code (reserved=249658KB, committed=2594KB)
 (malloc=58KB #348)
 (mmap: reserved=249600KB, committed=2536KB)
- GC (reserved=79628KB, committed=79544KB)
 (malloc=5772KB #118)
 (mmap: reserved=73856KB, committed=73772KB)
- Compiler (reserved=138KB, committed=138KB)
 (malloc=8KB #41)
 (arena=131KB #3)
- Internal (reserved=5380KB, committed=5380KB)
 (malloc=5316KB #1357)
 (mmap: reserved=64KB, committed=64KB)
- Symbol (reserved=1367KB, committed=1367KB)
 (malloc=911KB #112)
 (arena=456KB #1)
- Native Memory Tracking (reserved=118KB, committed=118KB)
 (malloc=66KB #1040)
 (tracking overhead=52KB)
- Arena Chunk (reserved=217KB, committed=217KB)
 (malloc=217KB)

Native Memory Leaks Outside JVM

- For the native memory leaks stemming from outside the JVM, we need to rely on the native memory leak tools for their detection and troubleshooting
- Native Memory Leak Detection Tools
 - dbx
 - libumem
 - valgrind
 - purify
 - and so on

Summary

- Causes of memory related problems:
 - Misconfiguration of Memory Pools
 - Excessive use of Finalizers
 - Explicit GC Invocations
 - Memory Leaks
- Size memory pools appropriately
- Tools
 - HeapDumpOnOutOfMemoryError and PrintClassHistogram JVM Options
 - JConsole
 - jcmd
 - jmap
 - GC Logs
 - Eclipse MAT
 - NMT
 - Native Memory Leak Detection Tools such as dbx, libumem, valgrind, purify etc.

References

- Troubleshooting Guide:
 - <https://docs.oracle.com/javase/9/troubleshoot/toc.htm>
 - <https://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot/index.html>
- Blog:
 - <https://blogs.oracle.com/poonam/>
- Free Online Course: Java Virtual Machine Troubleshooting
 - <http://www.oracle.com/goto/jvm>
- Article: <https://www.infoq.com/articles/Troubleshooting-Java-Memory-Issues>

Stay connected

- Join us: **DevOps Corner** (Developer Lounge – Moscone West)
- Learn more: **openjdk.java.net | wercker.com/java**
- Follow: **[@OpenJDK](#), [@wercker](#) [#JavaOne](#) [#DevOps](#)**



JavaOne™

ORACLE®