



Java™  
ORACLE®

# Lesson 3

## Diagnostic Data Collection and Analysis tools

Poonam Parhar  
JVM Sustaining Engineer  
Oracle

Java  
Your  
Next  
(Cloud)

# Agenda

Diagnostic Data, Data Collection and Analysis tools

1. Java Heap Memory Issues
2. OutOfMemoryError due to Finalization
3. PermGen/Metaspace Memory Issues
4. CodeCache Issues
5. Native Memory Issues

# Java Heap: Memory Leak

# 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

# Monitor using GC Logs

[GC (Allocation Failure) [PSYoungGen: 318596K->153251K(433152K)] 1184491K->1182018K(1556992K), 0.5548358 secs] [Times: user=1.78 sys=0.13, real=0.56 secs]

[Full GC (Ergonomics) [PSYoungGen: 153251K->0K(433152K)] [ParOldGen: 1028766K->1054946K(1345024K)] 1182018K->1054946K(1778176K), [Metaspace: 2722K->2722K(1056768K)], 4.5281743 secs] [Times: user=10.09 sys=0.00, real=4.52 secs]

[GC (Allocation Failure) [PSYoungGen: 209408K->209511K(448512K)] 1264354K->1264458K(1793536K), 0.1590964 secs] [Times: user=0.48 sys=0.06, real=0.15 secs]

[GC (Allocation Failure) [PSYoungGen: 434279K->223744K(448512K)] 1489226K->1489490K(1793536K), 0.4988033 secs] [Times: user=1.62 sys=0.26, real=0.49 secs]

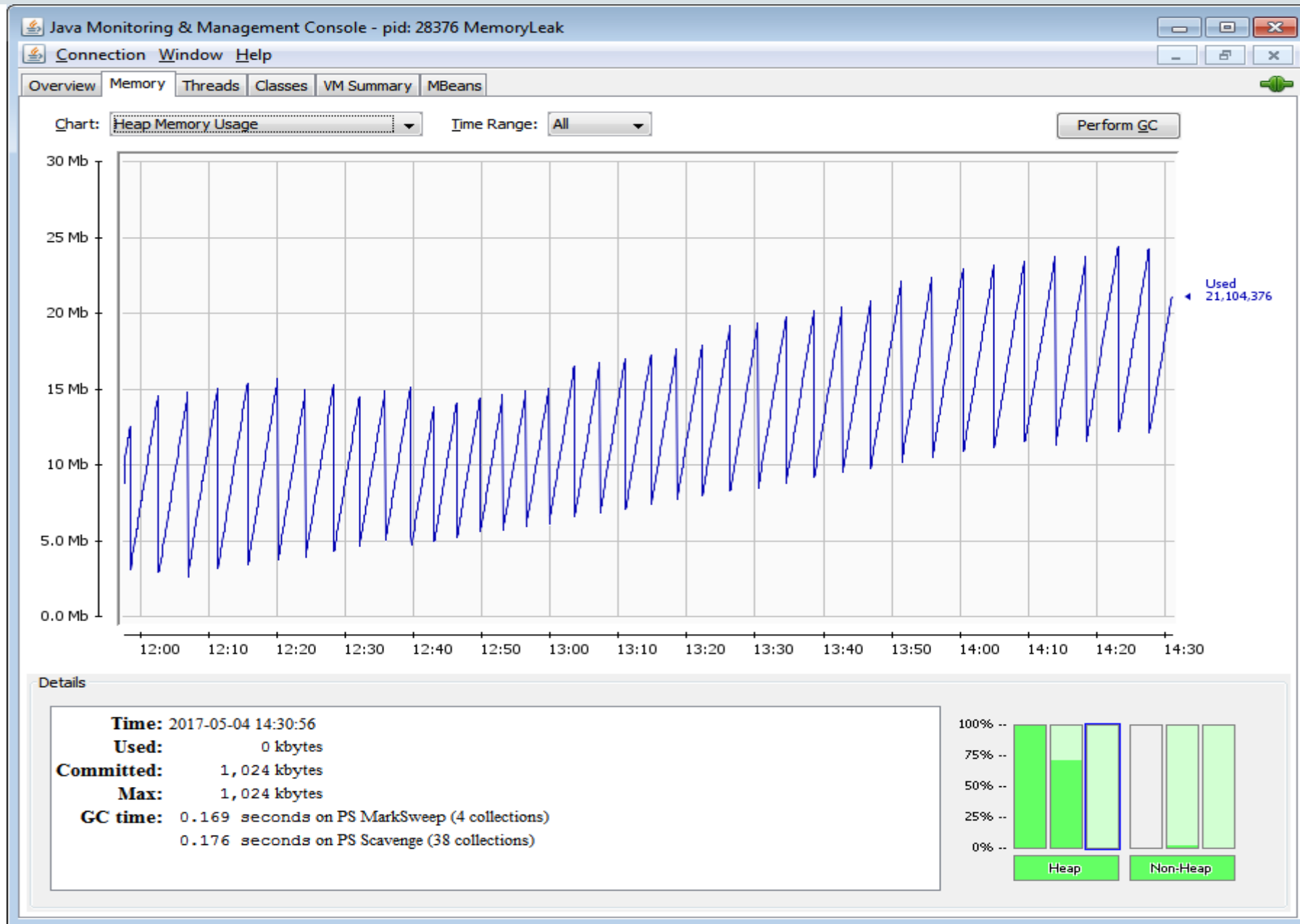
[Full GC (Ergonomics) [PSYoungGen: 223744K->36309K(448512K)] **[ParOldGen: 1265746K->1344646K(1345024K)]** 1489490K->1380956K(1793536K), [Metaspace: 2722K->2722K(1056768K)], 5.0727511 secs] [Times: user=12.65 sys=0.02, real=5.08 secs]

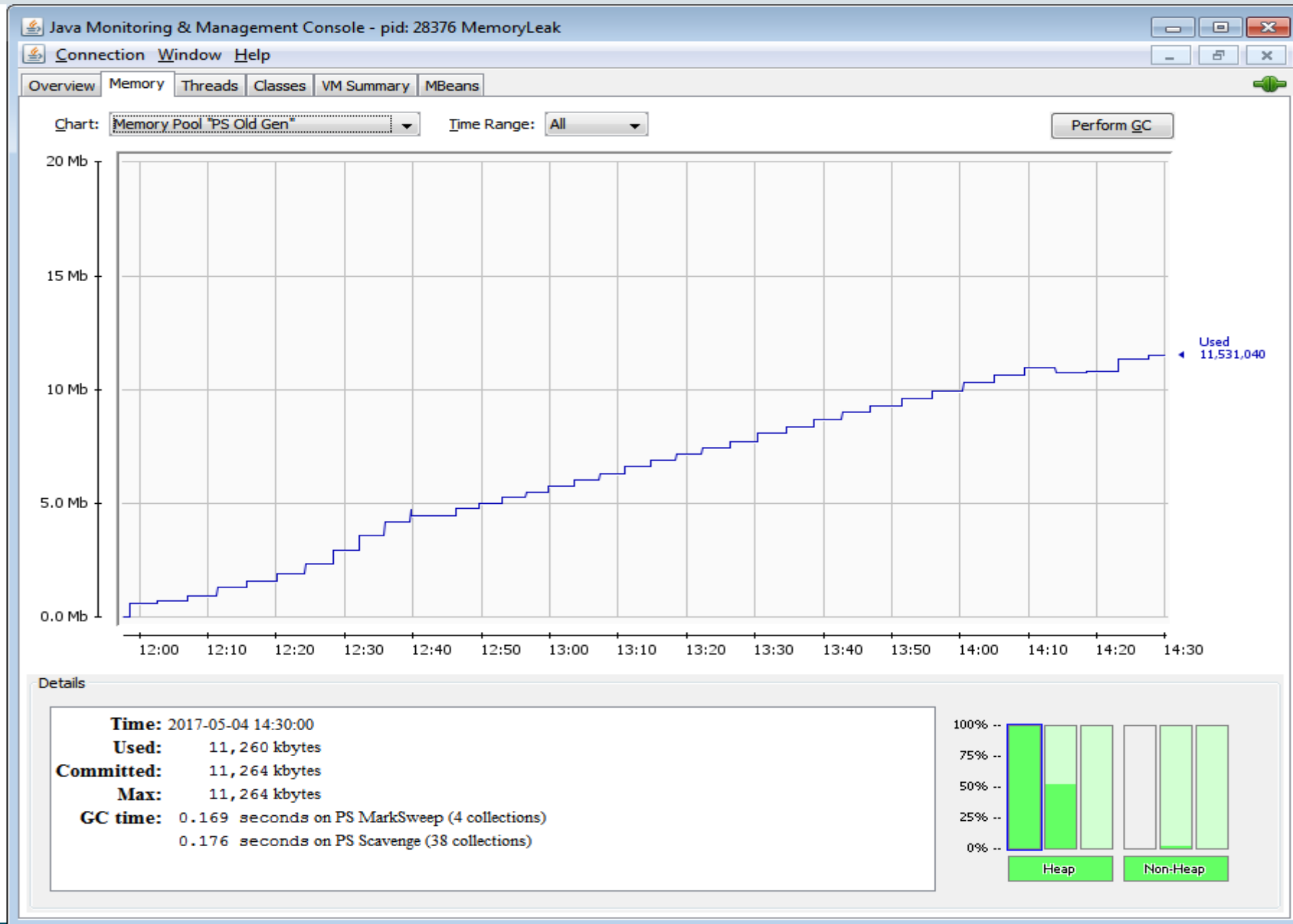
[Full GC (Ergonomics) [PSYoungGen: 197281K->197043K(448512K)] **[ParOldGen: 1344646K->1344646K(1345024K)]** 1541927K->1541690K(1793536K), [Metaspace: 2722K->2722K(1056768K)], 3.0359889 secs] [Times:user=11.82 sys=0.00, real=3.03 secs]

[Full GC (Allocation Failure) [PSYoungGen: 197043K->197043K(448512K)] **[ParOldGen: 1344646K->1344634K(1345024K)]** 1541690K->1541677K(1793536K), [Metaspace: 2722K->2722K(1056768K)], 6.9535358 secs][Times: user=20.80 sys=0.01, real=6.95 secs]

java.lang.OutOfMemoryError: Java heap space









# Appropriate Heap Size

**-Xmx**

# Java Heap: Diagnostic Data

# Java heap: Diagnostic Data

- GC Logs
  - Heap usage details
  - GC pauses
  - Help in appropriate configuration of memory pools
- Heap Dumps
  - Unexpected memory growth and memory leaks
- Heap Histograms
  - Quick view of the heap to understand what is growing
- Java Flight Recordings
  - Unexpected memory growth and memory leaks
  - GC Events

# 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 Mbean HotSpotDiagnostic
  - **-XX:+HeapDumpOnOutOfMemoryError**

Java Monitoring & Management Console - pid: 30584 MemoryLeak

Connection Window Help

Overview Memory Threads Classes VM Summary MBeans

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

Operation invocation

void dumpHeap p0 String , p1 true )

MBeanOperationInfo

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

Descriptor

Name	Value
<b>Operation:</b>	
openType	javax.management.openmbean.SimpleType(name=java.lang.Void)
originalType	void
<b>Parameter-0:</b>	
openType	javax.management.openmbean.SimpleType(name=java.lang.String)
originalType	java.lang.String
<b>Parameter-1:</b>	
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
  - **jmap** -histo <java> core\_file
  - **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

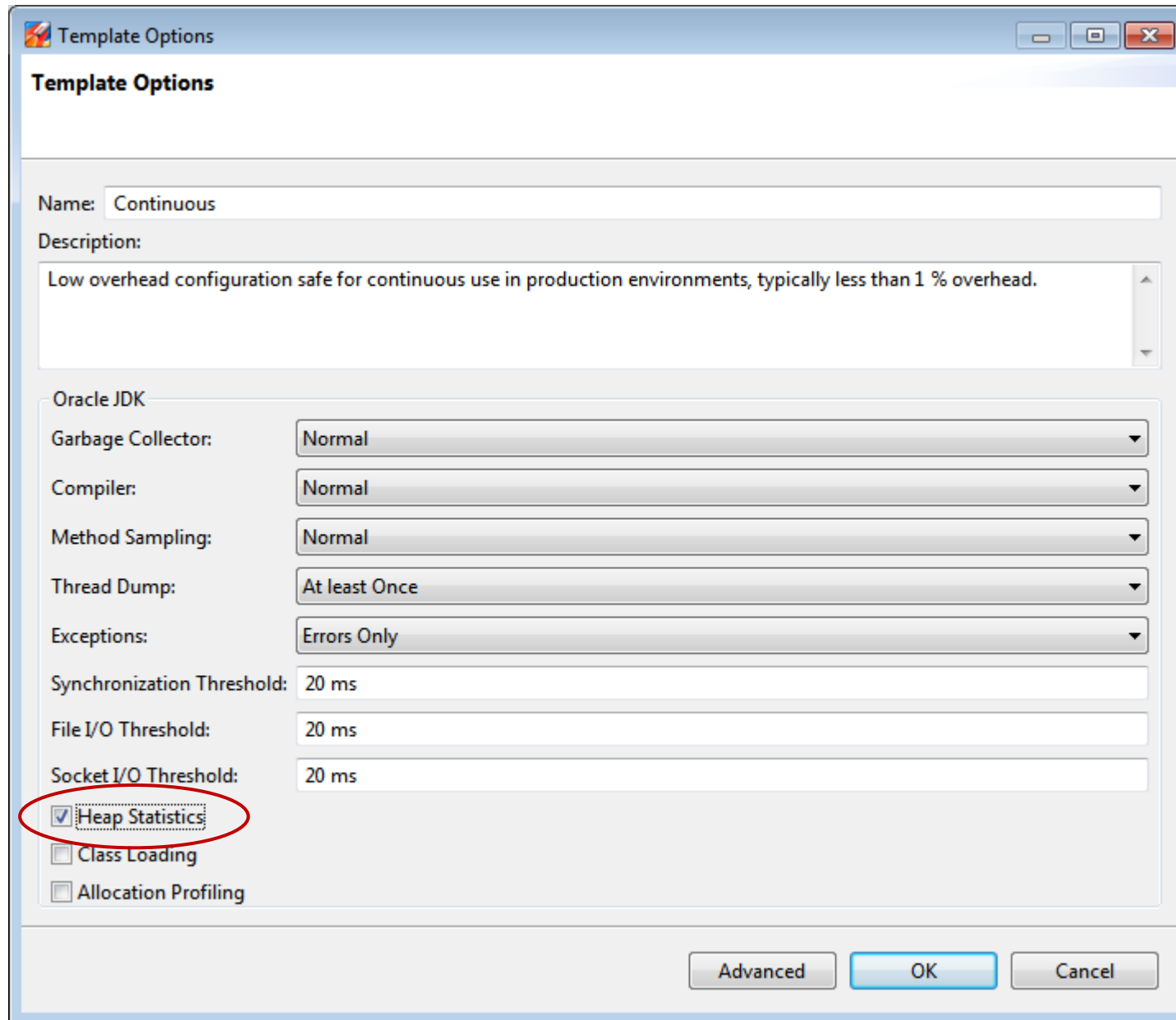
Overview MBean Browser Triggers System Memory Threads Diagnostic Commands

# Java Flight Recordings

- Flight Recordings with **Heap Statistics** enabled can be really helpful in troubleshooting memory leaks
- Enable 'Heap Statistics'
  - by going to 'Window->Flight Recording Template Manager' in JMC
  - Edit manually in the .jfc file:

```
<event path="vm/gc/detailed/object_count">  
    <setting name="enabled" control="heap-statistics-enabled">true</setting>  
    <setting name="period">everyChunk</setting>  
</event>
```





# Create Java Flight Recordings

- JVM Flight Recorder options, e.g.

```
-XX:+UnlockCommercialFeatures -XX:+FlightRecorder -  
  XX:StartFlightRecording=delay=20s,duration=60s,name=MyRecording,filename=C:\TEMP\myrecording.  
  jfr,settings=profile
```

- Java Diagnostic Command: jcmd

```
jcmd 7060 JFR.start name=MyRecording settings=profile delay=20s duration=2m  
  filename=c:\TEMP\myrecording.jfr
```

- Java Mission Control

– Connect to the process and follow the wizard

- The Flight Recordings can take us as far as determining the type of objects that are leaking but to find out what is causing those objects to leak, we require heap dumps

# 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.



1000%

file:D:/tests/gc\_logs.txt

Chart Event details

## Gc pauses

name	n	min (s)	max (s)	avg (s)	stddev	sum (s)	sum (%)
GC cleanup	1995	0.03218	0.22383	0.05467	0.01437	109.06035	0.8
GC pause (mixed)	1689	0.50019	2.44630	1.02430	0.33984	1,730.04518	12.9
GC pause (mixed) (to-space exhausted)	2	1.81449	2.43525	2.12487	0.43894	4.24974	0.0
GC pause (young)	5025	0.12226	2.47535	1.01099	0.35720	5,080.20377	37.9
GC pause (young) (initial-mark)	1997	0.38357	6.70860	2.77117	1.08715	5,534.03362	41.2
GC pause (young) (to-space exhausted)	2	2.85047	3.53174	3.19110	0.48173	6.38221	0.0
GC remark; GC ref-proc	1995	0.18893	1.39511	0.47750	0.17867	952.61412	7.1
total	12705					13,416.58899	98.5

## Full gc pauses

name	n	min (s)	max (s)	avg (s)	stddev	sum (s)	sum (%)
Full GC; Eden	2	73.16339	131.28775	102.22557	41.10013	204.45115	100.0
total	2					204.45115	1.5

## Concurrent GCs

name	n	min (s)	max (s)	avg (s)	stddev	sum (s)	sum (%)
GC concurrent-cleanup-end	33	0.00009	0.00236	0.00021	0.00040	0.00683	0.0
GC concurrent-cleanup-start	33	0.00000	0.00000	0.00000	0.00000	0.00000	0.0
GC concurrent-mark-abort	2	0.00000	0.00000	0.00000	0.00000	0.00000	0.0
GC concurrent-mark-end	1995	9.29759	37.49595	17.55920	4.88878	35,030.59919	99.6
GC concurrent-mark-start	1997	0.00000	0.00000	0.00000	0.00000	0.00000	0.0
GC concurrent-root-region-scan-end	1997	0.00006	0.68069	0.07749	0.03391	154.74334	0.4
GC concurrent-root-region-scan-start	1997	0.00000	0.00000	0.00000	0.00000	0.00000	0.0
total	8054					35,185.34937	

Summary Memory Pause

Total heap (usage / alloc. max)	18,534.4M (100.0%) / 18,534.4M
Tenured heap (usage / alloc. max)	18,534.4M (105.2%) / 17,614.4M
Young heap (usage / alloc. max)	11,092.8M (100.0%) / 11,092.8M
Perm heap (usage / alloc. max)	n/a
Max tenured after conc GC	15,976M (90.7% / 86.2%)
Avg tenured after conc GC	10,714.7M ( $\sigma=2,052.541M$ )
Max heap after conc GC	16,896M (91.2%)
Avg heap after conc GC	15,404.1M ( $\sigma=1,559.439M$ )
Max heap after full GC	7,343.2M (39.6%)
Avg heap after full GC	7,335.1M ( $\sigma=11.385M$ )
Avg after GC	12,450.1M ( $\sigma=5,668.947M$ )
Freed by full GC	22,398.5M (0.2%)
Freed by GC	10,491,881.2M (99.8%)
Avg freed full GC	11,199.3M/coll ( $\sigma=11.385M$ )
Avg freed GC	825.8M/coll ( $\sigma=1,186.443M$ )
Avg rel inc after FGC	-16,882,688B/coll
Avg rel inc after GC	936.049K/coll
Slope full GC	-127B/s
Slope GC	61.495K/s
InitiatingOccFraction (avg / max)	87.3% / 100.0%
Avg promotion	3,353.946K/coll ( $\sigma=53.722M$ )
Total promotion	28,544.568M

(to-space exhausted), 2.8504662 secs]

[Parallel Time: 2778.5 ms, GC Workers: 16]

[GC Worker Start (ms): Min: 122158804.8, Avg: 122158805.1, Max: 122158805.3, Diff: 0.5]

[Ext Root Scanning (ms): Min: 869.1, Avg: 896.0, Max: 952.5, Diff: 83.4, Sum: 14335.3]

[Update RS (ms): Min: 18.4, Avg: 27.0, Max: 34.6, Diff: 16.2, Sum: 431.5]

[Processed Buffers: Min: 18, Avg: 33.0, Max: 48, Diff: 30, Sum: 528]

[Scan RS (ms): Min: 0.0, Avg: 0.0, Max: 0.1, Diff: 0.1, Sum: 0.3]

[Code Root Scanning (ms): Min: 0.0, Avg: 0.0, Max: 0.0, Diff: 0.0, Sum: 0.0]

[Object Copy (ms): Min: 1805.5, Avg: 1854.5, Max: 1878.0, Diff: 72.5, Sum: 29671.2]

[Termination (ms): Min: 0.0, Avg: 0.2, Max: 0.3, Diff: 0.3, Sum: 3.0]

[GC Worker Other (ms): Min: 0.0, Avg: 0.2, Max: 0.4, Diff: 0.3, Sum: 2.8]

[GC Worker Total (ms): Min: 2777.4, Avg: 2777.8, Max: 2778.0, Diff: 0.6, Sum: 44444.1]

[GC Worker End (ms): Min: 122161582.7, Avg: 122161582.8, Max: 122161583.0, Diff: 0.3]

[Code Root Fixup: 8.4 ms]

[Code Root Migration: 0.0 ms]

[Clear CT: 0.5 ms]

[Other: 63.0 ms]

[Choose CSet: 0.0 ms]

[Ref Proc: 1.1 ms]

[Ref Enq: 0.0 ms]

[Free CSet: 0.4 ms]

[Eden: 64.0M(792.0M)->0.0B(920.0M) Survivors: 128.0M->0.0B Heap: 18.1G(18.1G)->12.1G(18.1G)]

[Times: user=25.31 sys=1.01, real=2.85 secs]

# 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*
- **Solution:** *-XX:+DisableExplicitGC*
- *kill -3* with *-XX:+PrintClassHistogram*



# 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

# JOverflow for Java Mission Control

- JOverflow is an experimental plugin
- Enables Java Mission Control to perform simple heap dump analysis and reports where the memory might have been wasted

# Java VisualVM

- All-in-one tool for monitoring, profiling and troubleshooting Java applications
- Available as a JDK tool as well as can be downloaded from GitHub
- Capable of doing heap dump analysis

# jhat

- Self-contained web application that is started at the command line (in our <jdk>/bin folder.)
- Enables heap dump analysis by browsing objects in the heap dump using any web browser
  - By default the web server is started at port 7000.
- jhat supports a wide range of pre-designed queries and Object Query Language(OQL) to explore the objects in the heap dumps
- Removed in JDK 9

# YourKit

- Commercial Java profiler with a heap dump analyzer
- YourKit offers Reachability Scope
- Memory Inspections
  - It offers a comprehensive set of built-in queries that can inspect the memory looking for anti-patterns and provides causes and solutions for common memory problems

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 >>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	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'l...	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.Obj...	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>b byte[1024] @ 0xfe7bd378</b>	<b>1</b>	<b>1,040</b>	<b>1,040</b>	<b>1,040</b>

735M of 1171M

# Java Flight Recording Analysis

# Java Mission Control

- Java Mission Control is available in the <jdk>/bin folder of the JDK.
- Flight Recordings collected with Heap Statistics enabled can greatly help in troubleshooting memory leaks
- Object Statistics under Memory->Object Statistics.
  - Shows the object histogram including the percentage of the heap that each object type occupies
  - Shows Top Growers in the heap. These usually have a direct correlation with the leaking objects

Oracle Java Mission Control

flight\_recording\_180152eaMemoryLeak23336.jfr

### Object Statistics

Interval: 59 s 893 ms (all)  Synchronize Selection

5/8/17 9:46:01 AM 5/8/17 9:47:01 AM

#### Heap Contents

Shows classes that take up more than 0.5% of the heap. If multiple snapshots are selected an average is shown.

Filter Column: Class

Class	Instances	Size	Percentage of Heap
char[]	9,462	689.61 kB	33.17%
byte[]	633	351.20 kB	16.89%
java.lang.Class	2,773	308.07 kB	14.82%
java.lang.String	9,424	220.86 kB	10.62%
java.lang.Object[]	2,098	115.80 kB	5.57%

#### Top Growers

Shows the increase between the first and last snapshot in the range.

Filter Column: Class

Class	Instance Increase	Size Increase
byte[]	64	76.03 kB
java.util.TreeMap\$Entry	188	7.34 kB
char[]	45	3.16 kB
java.lang.String	45	1.05 kB
java.util.LinkedHashMap\$Entry	24	960 bytes

Overview | Garbage Collections | GC Times | GC Configuration | Allocations | Object Statistics

# 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

# 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

---

<b>Connection name:</b> pid: 23336 MemoryLeak	<b>Uptime:</b> 14 minutes
<b>Virtual Machine:</b> Java HotSpot(TM) 64-Bit Server VM version 25.152-b01	<b>Process CPU time:</b> 6.318 seconds
<b>Vendor:</b> Oracle Corporation	<b>JIT compiler:</b> HotSpot 64-Bit Tiered Compilers
<b>Name:</b> 23336@POBAJAJ-LAP	<b>Total compile time:</b> 3.190 seconds

---

<b>Live threads:</b> 15	<b>Current classes loaded:</b> 2,593
<b>Peak:</b> 15	<b>Total classes loaded:</b> 2,716
<b>Daemon threads:</b> 14	<b>Total classes unloaded:</b> 123
<b>Total threads started:</b> 21	

---

<b>Current heap size:</b> 11,393 kbytes	<b>Committed memory:</b> 25,088 kbytes
<b>Maximum heap size:</b> 25,088 kbytes	<b>Pending finalization:</b> 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

---

<b>Operating System:</b> Windows 7 6.1	<b>Total physical memory:</b> 8,068,736 kbytes
<b>Architecture:</b> amd64	<b>Free physical memory:</b> 2,623,304 kbytes
<b>Number of processors:</b> 4	<b>Total swap space:</b> 16,135,612 kbytes
<b>Committed virtual memory:</b> 102,012 kbytes	<b>Free swap space:</b> 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
boole...	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
boole...	stillborn	false
boole...	daemon	true
boole...	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

# Finalization Info from Heap Dump with VisualVM

Java VisualVM

File Applications View Tools Window Help

Applications [heapdump] 29412.jmap

Local  
JConsole (pid 9724)  
MemoryUsage (pid 12760)  
VisualVM  
Remote  
Snapshots

[heapdump] 29412.jmap

Heap Dump

Summary Classes Instances OQL Console

Overview

**Basic info:**

- Date taken: Mon Feb 27 08:30:18 PST 2017
- File: D:\support\_data\_2\amy\29412.jmap
- File size: 339.7 MB
- Total bytes: 298,985,704
- Total classes: 1,554
- Total instances: 7,927,412
- Classloaders: 5
- GC roots: 1,039
- Number of objects pending for finalization: 0

**Environment:**

- OS: Linux (2.6.18-417.0.0.0.1.el5)
- Architecture: amd64 64bit
- Java Home: /apps/applmgr/r12/fs2/EBSapps/comn/util/jdk64/jre
- Java Version: 1.7.0\_25
- JVM: Java HotSpot(TM) 64-Bit Server VM (23.25-b01, mixed mode)
- Java Vendor: Oracle Corporation

**System properties:**

- JavaHome=/apps/applmgr/r12/fs2/EBSapps/comn/util/jdk64
- ListenAddress=bxv1116.appl.ge.com
- ListenPort=5557
- LogAppend=false
- NodeManagerHome=/apps/applmgr/r12/fs2/FMW\_Home/wlserver\_10.3/common/nodemanager/nmHome1
- QuitEnabled=true
- SecureListener=false
- StartScriptEnabled=false
- StartScriptName=startWebLogic.sh
- awt.toolkit=sun.awt.X11.XToolkit
- file.encoding=UTF-8
- file.encoding.pkg=sun.io
- file.separator=/
- java.awt.graphicsenv=sun.awt.X11GraphicsEnvironment
- java.awt.printerjob=sun.print.PSPrinterJob

Inspect

Biggest objects:

Find 20 biggest objects by retained size: Find

Computing retained sizes...

# PermGen/MetaSpace: Memory leak

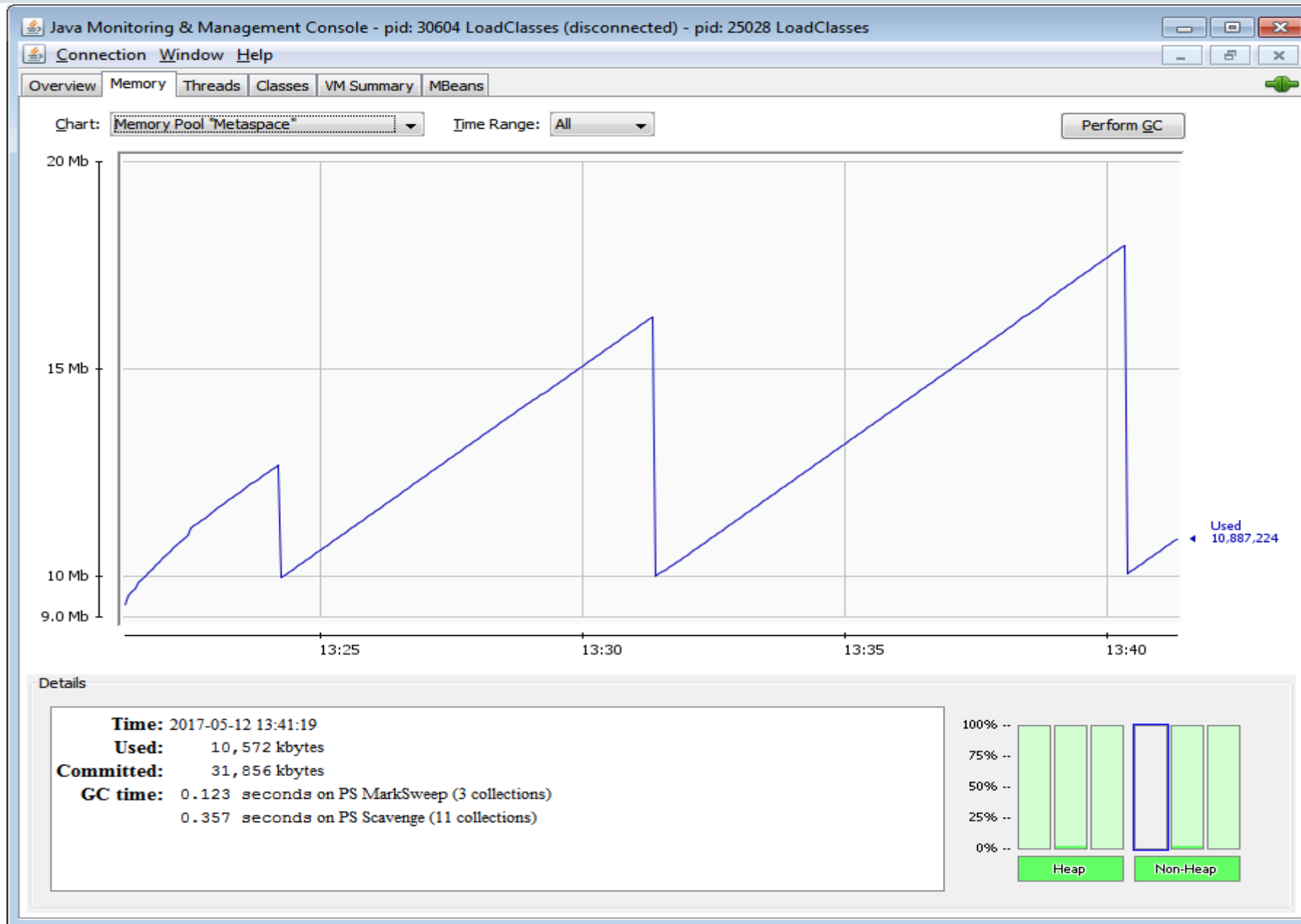
# Confirm Memory Leak

- 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

# Monitor GC Logs

166687.013: [Full GC [PSYoungGen: 126501K->0K(922048K)] [PSOldGen: 2063794K->1598637K(2097152K)] 2190295K->1598637K(3019200K) [PSPermGen: 165840K->164249K(166016K)], 6.8204928 secs] [Times: user=6.80 sys=0.02, real=6.81 secs]

166699.015: [Full GC [PSYoungGen: 125518K->0K(922048K)] [PSOldGen: 1763798K->1583621K(2097152K)] 1889316K->1583621K(3019200K) [PSPermGen: 165868K->164849K(166016K)], 4.8204928 secs] [Times: user=4.80 sys=0.02, real=4.81 secs]



# 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

# 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'`
- Java Flight Recordings

# 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$RBClassLoader@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/redeployments, 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.Prc
ref	classes	java.util.Vector
ref	package2certs	java.util.Hashtal
ref	parallelLockMap	null
ref	parent	weblogic.util.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
<b>Total: 9 entries</b>			
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
<b>Total: 9 entries</b>			
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

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 allocations 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 memory leak
  - Example: JNI code allocating native buffers

# Native OutOfMemoryError: Common Issues

# Native Heap OutOfMemoryError with 64-bit JVM

- Running with 32-bit JVM puts a maximum limit of 4GB on the process size
  - So you're more likely to run out of native memory with a 32-bit Java process
- Running with a 64-bit JVM gets us access to unlimited address space, so we would expect never to run out of native memory
- However, we might see OutOfMemoryErrors occurring in a 64-bit JVM too
- CompressedOops feature implementation determines where the Java heap should be placed in the address space
- The position of the Java heap can put a cap on the maximum size of the native heap.

# Memory Map

```
0000000100000000 8K r-x-- /sw/.es-base/sparc/pkg/jdk-1.7.0_60/bin/sparcv9/java
0000000100100000 8K rwx-- /sw/.es-base/sparc/pkg/jdk-1.7.0_60/bin/sparcv9/java
0000000100102000 56K rwx-- [ heap ]
0000000100110000 2624K rwx-- [ heap ] <--- native Heap
00000001FB000000 24576K rw--- [ anon ] <--- Java Heap starts here
0000000200000000 1396736K rw--- [ anon ]
0000000600000000 700416K rw--- [ anon ]
```

# Solution for OutOfMemoryError with 64-bit JVM

- Can be resolved by using option `-XX:HeapBaseMinAddress=n` to specify the address the Java heap should be based at
- Setting it to a higher address would leave more room for the native heap

# JAXB Issues

- JAXB internally uses Inflater/Deflater to compress/uncompress files
  - Inflater/Deflater use native memory to hold their data
  - Depend on Finalization to deallocate the java objects and the associated native memory data
  - Delay in running Finalizer can exhaust native memory
- `JAXBContext.newInstance()` called for every new request
  - Classes from the context get reloaded again
  - Increases native memory usage
- Environment upgrade fails to upgrade all the JAXB jar files
  - Classes linking errors leading to re-loading of classes

# OutOfMemoryError: Direct buffer memory

- `ByteBuffer.allocateDirect(SIZE_OF_BUFFER)`
- DirectByteBuffers are garbage collected by using a phantom reference and a reference queue
- Maximum direct memory is unbounded by default, but can be limited by using JVM option `-XX:MaxDirectMemorySize=n`

# NIO ByteBuffer

- Java NIO APIs use ByteBuffer as the source and destination of I/O calls
  - Java Heap ByteBuffer and Native Heap ByteBuffer
  - Java Heap ByteBuffer for I/O use a temporary direct ByteBuffer per thread
  - If large heap ByteBuffer from multiple threads are used for I/O calls
    - Native Heap Exhaustion
- `-Djdk.nio.maxCachedBufferSize=m` (JDK9)



# 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`

```
jcmm 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

- Several kinds of `OutOfMemoryError` messages
- It is important to understand these error messages clearly
- Tools
  - `HeapDumpOnOutOfMemoryError` and `PrintClassHistogram` JVM Options
  - Eclipse MAT
  - VisualVM
  - JConsole
  - `jhat`
  - YourKit
  - `jmap`
  - `jcmt`
  - Java Flight Recorder and Java Mission Control
  - GC Logs
  - 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>





Java™  
ORACLE®