ORACLE®

Java™

ORACLE®

Troubleshooting Memory Problems in Java Applications

Bangalore JUG

Fairoz Matte
Principle Member Of Technical Staff
Java Developer Community Support Engineer
Dec 16, 2017

Java Your Next (Cloud)

Safe Harbor Statement

Agenda

Agenda

1. Symptoms of memory related issues
2. Step 1: Understand the underlying problem
3. Step 2: Collect the Diagnostic data
4. Step 3: Analyze the data to find the issue
5. Demo examples

Symptoms of memory related issues

- Application Failure with java.lang.OutOfMemoryError
- Unexpected Memory growth
- Poor Application performance

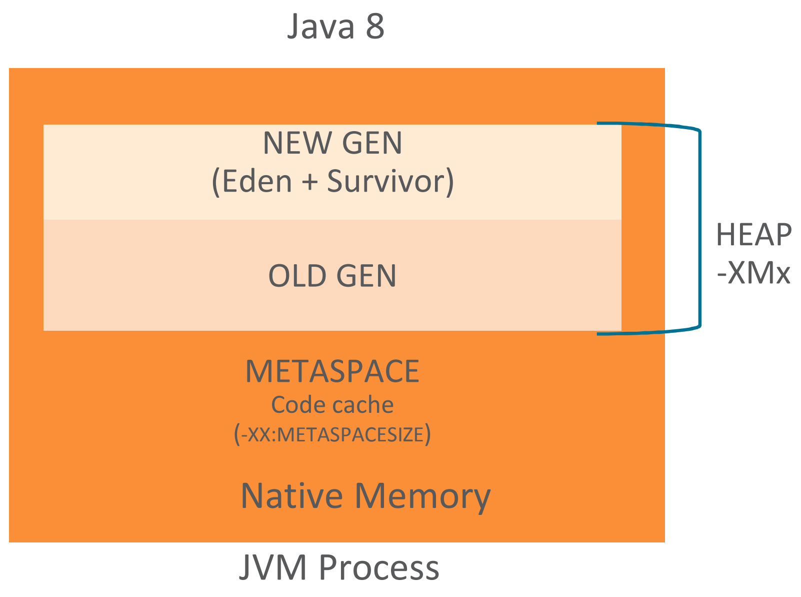
- GC activities
- Long GC pauses
- Excessive GC's

Agenda

1. Understand the underlying problem

- Misconfiguration of memory pools
- Excessive use of Finalizers
- Explicit GC invocation
- Memory leaks
- User Errors

1. Understand the underlying problem



1. Understand the underlying problem

Causes of java.lang.OutOfMemoryError

- java.lang.OutOfMemoryError: Java heap space
- java.lang.OutOfMemoryError: GC overhead limit exceeded
- java.lang.OutOfMemoryError: Unable to create new native thread
- java.lang.OutOfMemoryError: Metaspace/Compressed class space
- java.lang.OutOfMemoryError: Out of swap space?
- java.lang.OutOfMemoryError: Requested array size exceeds VM limit

Agenda

2. Collect the Diagnostic data

- **GC logs** (-Xlog:gc, -XX:+PrintGCDetails,-XX:+PrintGCTimeStamps)
  - Very helpful in determining the heap requirements
  - Excessive GC's
  - Long GC Pauses
- **Heap dumps** (jmap, jcmd, jvisualvm, java mission control,-XX:+HeapDumpOnOutofMemoryError)
  - Most important data for analyzing any memory related issues
- **Heap histograms** (jmap, jcmd, jvisualvm, java mission control,-XX:+PrintClassHistogram, SIGQUIT,SIGBREAK)
  - Quick view on objects and allocation in heap

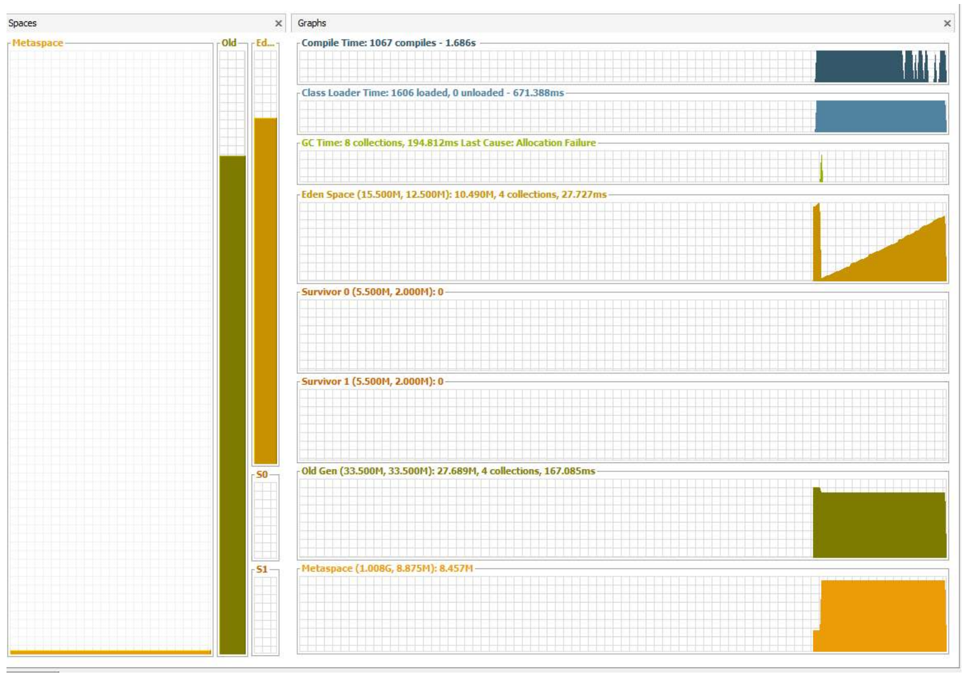
2. Collect the Diagnostic data

```
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]
```

```
[Full GC (Ergonomics) [PSYoungGen: 48341K->44325K(59904K)] [ParOldGen: 133231K->136249K(136704K)] 181572K->180574K(196608K),
[Metaspace: 2628K->2628K(1056768K)], 2.0914060 secs] [Times: user=5.54 sys=0.02, real=2.09 secs]
[Full GC (Allocation Failure) [PSYoungGen: 44325K->44325K(59904K)] [ParOldGen: 136249K->136235K(136704K)] 180574K-
>180560K(196608K), [Metaspace: 2628K->2628K(1056768K)], 1.1300763 secs] [Times: user=3.98 sys=0.00, real=1.13 secs]
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
at java.util.Arrays.copyOf(Arrays.java:3210)
at java.util.Arrays.copyOf(Arrays.java:3181)
at java.util.ArrayList.grow(ArrayList.java:261)
at java.util.ArrayList.ensureExplicitCapacity(ArrayList.java:235)
at java.util.ArrayList.ensureCapacityInternal(ArrayList.java:227)
at java.util.ArrayList.add(ArrayList.java:458)
at oom.Wrapper.main(Wrapper.java:12)
Heap
 PSYoungGen      total 59904K, used 46382K [0x00000000fbd80000, 0x0000000100000000, 0x0000000100000000)
  eden space 51712K, 89% used [0x00000000fbd80000,0x00000000feacba40,0x00000000ff000000)
  from space 8192K, 0% used [0x00000000ff000000,0x00000000ff000000,0x00000000ff800000)
  to   space 8192K, 0% used [0x00000000ff800000,0x00000000ff800000,0x0000000100000000)
 ParOldGen       total 136704K, used 136235K [0x00000000f3800000, 0x00000000fbd80000, 0x00000000fbd80000)
  object space 136704K, 99% used [0x00000000f3800000,0x00000000fbd0aeb0,0x00000000fbd80000)
 Metaspace       used 2660K, capacity 4486K, committed 4864K, reserved 1056768K
  class space    used 287K, capacity 386K, committed 512K, reserved 1048576K
```

2. Collect the Diagnostic data

Visual GC from Visual VM



Compile Time: 1067 compiles - 1.686s

Class Loader Time: 1606 loaded, 0 unloaded - 671.388ms

GC Time: 8 collections, 194.812ms Last Cause: Allocation Failure

Eden Space (15.500M, 12.500M): 10.490M, 4 collections, 27.727ms

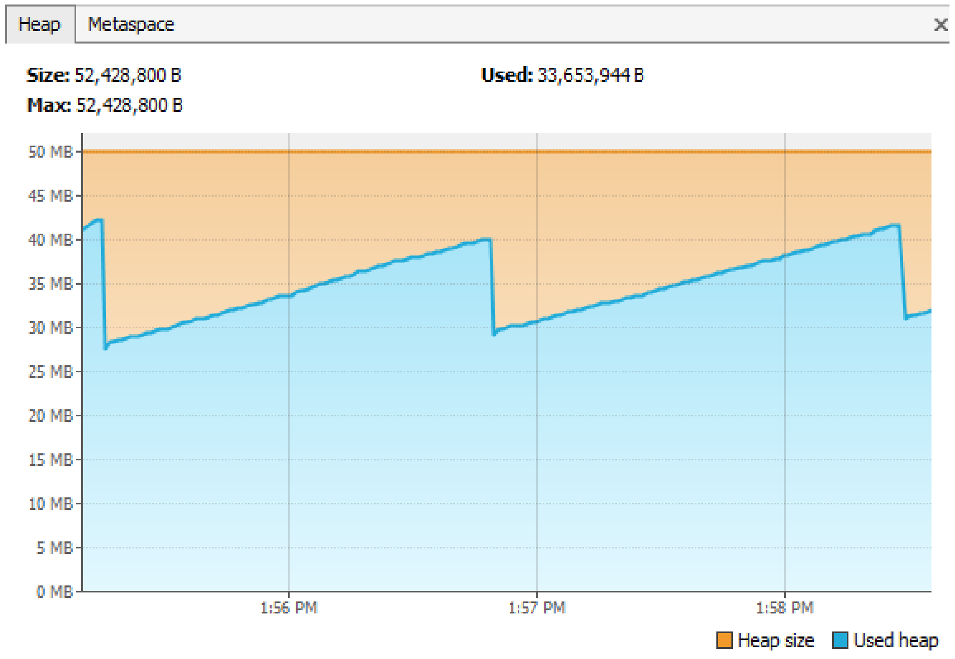
Survivor 0 (5.500M, 2.000M): 0

Survivor 1 (5.500M, 2.000M): 0

Old Gen (33.500M, 33.500M): 27.689M, 4 collections, 167.085ms

Metaspace (1.008G, 8.875M): 8.457M

Heap Allocation from Visual VM



Size: 52,428,800 B **Used:** 33,653,944 B
Max: 52,428,800 B

Agenda

3. Analyze the data to find the issue

Use the right tool to find the exact problem

Solution to common problems

1. Use Appropriate heap size using -Xmx option (Make sure you have enough memory)
2. Do not use explicit GC calls (System.gc())
3. Avoid finalizers (Deprecated - JDK-8165641)
4. Avoid user errors while handling memory operations/ opening closing file handles or any resources

Agenda

Demo 1: java.lang.OutOfMemoryError: Java heap space

```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at java.util.Arrays.copyOf(Arrays.java:3210)
        at java.util.Arrays.copyOf(Arrays.java:3181)
        at java.util.ArrayList.grow(ArrayList.java:261)
        at java.util.ArrayList.ensureExplicitCapacity(ArrayList.java:235)
        at java.util.ArrayList.ensureCapacityInternal(ArrayList.java:227)
        at java.util.ArrayList.add(ArrayList.java:458)
        at oom.Wrapper.main(Wrapper.java:12)
```

Demo 2: " java.lang.OutOfMemoryError: unable to create new native thread

```
Exception in thread "main" java.lang.OutOfMemoryError: unable to create new native thread
	at java.lang.Thread.start0(Native Method)
	at java.lang.Thread.start(Thread.java:717)
	at oom.GC_NativeOOM.main(GC_NativeOOM.java:13)
```

Demo 2: " java.lang.OutOfMemoryError: unable to create new native thread

Debug:
1. Use Task manager to track usage
2. Start application with -XX:NativeMemoryTracking=summary or -XX:NativeMemoryTracking=detail

Jcmd to monitor the NMT (native memory tracker)
Jcmd <pid> VM.native_memory baseline
Jcmd <pid> VM.native_memory detail.diff

Demo 3: " java.lang.OutOfMemoryError: GC overhead limit exceeded

```
Exception in thread "main" java.lang.OutOfMemoryError: GC overhead limit exceeded
Dec 11, 2017 8:58:11 PM sun.rmi.transport.tcp.TCPTransport$AcceptLoop executeAcceptLoop
WARNING: RMI TCP Accept-0: accept loop for ServerSocket[addr=0.0.0.0/0.0.0.0,localport=57280]
throws
java.lang.OutOfMemoryError: GC overhead limit exceeded
at java.net.NetworkInterface.getAll(Native Method)
at java.net.NetworkInterface.getNetworkInterfaces(NetworkInterface.java:343)
at
sun.management.jmxremote.LocalRMIServerSocketFactory$1.accept(LocalRMIServerSocketFactor
y.java:86)
at sun.rmi.transport.tcp.TCPTransport$AcceptLoop.executeAcceptLoop(TCPTransport.java:400)
at sun.rmi.transport.tcp.TCPTransport$AcceptLoop.run(TCPTransport.java:372)
at java.lang.Thread.run(Thread.java:748)
```

References

- https://www.youtube.com/watch?v=iixQAYnBnJw
- https://bugs.openjdk.java.net/browse/JDK-8165641
- https://www.ibm.com/developerworks/java/library/j-nativememory-linux/

Q&A

Java Your Next (Cloud)