Liberica JDK
Guide to JVM memory configuration options

Liberica JDK
Revision 1.0
April 2024

bellsoft

BellSoft software contains open source software. Additional information about third party code is available at https://bell-sw.com/third_party_licenses. You can also get more information on how to get a copy of source code by contacting info@bell-sw.com.

The use of any software product referenced in this document is governed by the applicable license agreement, which is not modified in any way by the terms of this notice.

Contents

1. Introduction

Reducing the memory footprint of the application requires meticulous optimizations with due consideration of all variables. This document contains an overview of the most important JVM flags related to memory management.

2. Heap size options

JVM parameter	Description
-Xms	Sets the initial heap size
-Xmx	Sets the maximum heap size
-XX:MinHeapFreeRatio	Sets the minimum percentage of free space after garbage collection
-XX:MaxHeapFreeRatio	Sets the maximum percentage of free space after garbage collection
-XX:MaxDirectMemorySize	Sets the limit for the memory allocated to direct byte buffers

In some cases, setting the maximum and minimum Java heap size is enough to optimize JVM memory footprint. The optimal heap size depends on your application, so you should experiment with the values before settling on a final number.

Setting min. and max. proportion of heap free after GC helps to avoid unnecessary expansion and shrinking of free space and release the unused memory without affecting the performance significantly.

For instance, if you set `-XX:MinHeapFreeRatio=40` and `-XX:MaxHeapFreeRatio=70`, then the generation expands if the free space percentage goes below 40% and contracts if the free space exceeds 70%.

Direct byte buffers are used by the JVM to perform native I/O operations. As opposed to non-direct byte buffers stored in the heap, direct ones reside outside the heap and therefore are not affected by heap size parameters or garbage collection. By default, the JVM chooses the size of the direct size buffers automatically based on the available memory, so setting the `-XX:MaxDirectMemorySize` helps to prevent excessive resource consumption.

3. RAM consumption

JVM parameter	Description
-XX:MaxRAM	Sets the max. amount of total memory used by the JVM
-XX:MaxRAMFraction	Sets the RAM limit for JVM in fractions
-XX:MaxRAMPercentage	Sets the RAM limit for JVM per cent

The JVM flags adjusting the heap size do not affect the total memory consumption by the JVM. To limit the total RAM consumption, use MaxRam flags. The heap size will be adjusted accordingly. For instance, if you have 1 GB of memory, setting `-XX:MaxRAMPercentage=50` (or `-XX:MaxRAMFraction=2`) will make the JVM allocate approx. 500 MB to heap.

These arguments are especially useful in the case of containerized applications, where they help to adjust the heap size based on the available container memory.

4. GC selection and logging

Selection

JVM parameter	Description
-XX:+UseSerialGC	Enables Serial Garbage Collector
-XX:+UseParallelGC	Enables Parallel Garbage Collector
-XX:+UseConcMarkSweepGC	Enables Concurrent Mark Sweep Garbage Collector (available up to Java 8 only)
-XX:+UseG1GC	Enables G1 Garbage Collector
-XX:+UseZGC (since Java 15)	Enables Z Garbage Collector (available since Java 11)
-XX:+UnlockExperimentalVMOptions -XX:+UseZGC (since Java 11 up to 15)	
-XX:+UseShenandoahGC	Enables Shenandoah Garbage Collector (absent in Oracle Java, available in major OpenJDK distributions)

The default garbage collection settings are enough for many applications. If you would like to enhance some KPIs (in this case, memory footprint), try switching to another collector, whose defaults are more beneficial to your app, without delving into the intricacies of GC tuning.

Java provides a set of GC implementations, each tailored to specific needs and use cases:

- Serial GC works in one thread and freezes all app threads while performing collection.
- Parallel GC also freezes all threads, but works in multiple threads itself.

- CMS GC does not freeze application threads, but instead, uses a few of them to perform its tasks. This collector was deprecated in Java 9 in favor of a more advanced G1 GC.
- G1 GC utilizes the Garbage-First approach by dividing the heap in areas and collects the garbage in mostly free areas thus releasing lots of memory.
- Z GC performs expensive work concurrently with the program and does not freeze the app threads for more than 10 ms.
- Shenandoah GC performs most of its work concurrently with the program, including the concurrent compaction, so the GC pause times are not directly proportional to the heap size.

Logging

JVM parameter	Description
-Xlog:gc*:<gc.log file path>:time	Stores the GC logging data at the specified location
-XX:PrintGC	Enables basic logging in Java 8
-XX:+PrintGCDetails	Activates detailed logging in Java 8+
-XX:NumberOfGCLogFiles	Sets the limit for the number of GC logs in Java 8
-XX:GCLogFileSize	Sets the max. size of a GC log file in Java 8

Before adjusting garbage collector settings, learn to understand its behavior. GC logs are text files that provide exhaustive information about GC work: total GC time, memory reclamation and allocation, etc.

Note that GC logging parameters vary between Java 8 and Java 9+:

- `-XX:+PrintGCDetails` and `-Xlog:gc` in Java 9+ substitute `-XX:PrintGC` in Java 8;
- Java 8 includes the `-XX:+UseGCLogFileRotation` parameter that enables the rotation of GC logs. It is used together with the `-XX:NumberOfGCLogFiles` and `-XX:GCLogFileSize` flags. However, these functions were deprecated in newer Java versions.

A new unified GC logging system is implemented with JEP 271. To learn more about the new logging

syntax, run:

```
-Xlog:help
```

5. GC management

JVM parameter	Description
-XX:GCTimeRatio	Sets the limit for GC execution time
-XX:AdaptiveSizePolicyWeight	Specifies how much previous GC times are taken into consideration when calculating current timing goals
-XX:+UseCGroupMemoryLimitForHeap	Sets the heap size based on the available container memory
-XX:ParallelGCThreads	Sets the number of Parallel GC threads
XX:G1HeapRegionSize	Sets the size of a G1 region
XX:InitiatingHeapOccupancyPercent	Sets the heap occupancy threshold triggering a marking cycle

Each GC comes with numerous settings that enable the developers to adjust latency, throughput, or memory. The table above provides memory related settings.

It should be noted that by default, `-XX:GCTimeRatio` is set to 99, which means that the application gets 99% of total execution time, and the collector can run for not more than 1% of the time. The `-XX:GCTimeRatio` and `-XX:AdaptiveSizePolicyWeight` parameters are helpful when using `-XX:MinHeapFreeRatio` and `-XX:MaxHeapFreeRatiowith` Parallel GC.

6. How to handle OutOfMemoryError

JVM parameter	Description
-XX:+HeapDumpOnOutOfMemoryError	Dumps heap into a file in the case of OutOfMemoryError
-XX:HeapDumpPath	Specifies the path for the file with heap data
-XX:OnOutOfMemoryError="< cmd args >;< cmd args >"	Specifies actions to be performed in the case of `OutOfMemoryError`

`OutOfMemoryError` leads to the application crash and is hard to troubleshoot. The above parameters provide the developers with a lot of information related to the error, so it is easier to detect memory leaks.

7. Working with Strings

JVM parameter	Description
-XX:+UseStringDeduplication	Removes duplicate strings during GC (with G1 GC only)
-XX:+UseStringCache	Caches commonly allocated strings in the String pool
-XX:+UseCompressedStrings	Uses a *byte[]* for Strings that can be represented as pure ASCII
-XX:+OptimizeStringConcat	Optimizes String concatenation operations when possible

`java.lang.String` is the most commonly used Java class. No wonder that Strings take up a significant part of the application memory. We can release the resources by removing duplicate strings and optimizing the String operations with the above parameters.

8. Other useful parameters

JVM parameter	Description
-XX:LargePageHeapSizeThreshold	Uses large pages if max. heap is at least as big as the specified value
-XX:LargePageSizeInBytes	Sets the large page size for the heap
XX:+UseCompressedOops	Enables the use of compressed pointers (32-bit instead of 64-bit) for heaps less than 32 GB
-XX:+TieredCompilation	Disables intermediate compilation tiers
-XX:TieredStopAtLevel=1	Uses only the C1 compiler
-XX:ThreadStackSize	Sets the size of thread stack space

The `-XX:LargePageHeapSizeThreshold` and `-XX:LargePageSizeInBytes` flags enable the developers to operate with large pages (a technique to reduce the pressure on the processors Translation-Lookaside Buffer caches) and make better use of virtual hardware resources.

The `-XX:+TieredCompilation` and `-XX:TieredStopAtLevel=1` can be used with Serial GC to turn off the optimizing compiler and reduce memory footprint in some cases. Use them when memory consumption is the only important KPI.

Memory to thread stacks is allocated outside the heap, so it is not affected by heap size parameters. The `-XX:ThreadStackSize` flag enables the developers to reduce the size of thread stacks.

9. Conclusion

There are a few more JVM options left unmentioned in this document, such as the ones adjusting the size of different heap spaces (permanent generation, young generation, Eden, survivor). The reason is that these parameters require extremely fine-tuning without significant overall improvement of memory consumption.

Liberica JDK
Guide to JVM memory configuration options

bell//soft