(http://baeldung.com)

Guide to the Most Important JVM Parameters

Last modified: April 15, 2018

by baeldung (http://www.baeldung.com/author/baeldung/)

Java (http://www.baeldung.com/category/java/) +

1. Overview

In this quick tutorial, we'll explore the most well-known options which can be used to configure the Java Virtual Machine.

2. Explicit Heap Memory

One of the most common performance-related practices is to initialize the heap memory as per the application requirements.

That's why we should specify minimal and maximal heap size. Below parameters can be used for achieving it:

```
-Xms<heap size>[unit]
-Xmx<heap size>[unit]
```

Here, ***unit*** denotes the unit in which the memory (indicated by ***heap size***) is to be initialized. Units can be marked as ***'g'*** for GB, ***'m'*** for MB and ***'k'*** for KB.

For example, if we want to assign minimum 2 GB and maximum 5 GB to JVM, we need to write:

```
-Xms2G -Xmx5G
```

Starting with Java 8, the size of *Metaspace (https://blogs.oracle.com/poonam/about-g1-garbage-collector,-permanent-generation-and-metaspace)* is not defined. Once it reaches the global limit, JVM automatically increases it, However, to overcome any unnecessary instability, we can set *Metaspace* size with:

```
-XX:MaxMetaspaceSize=<metaspace size>[unit]
```

Here, ***metaspace size*** denotes the amount of memory we want to assign to *Metaspace*.

As per Oracle guidelines (https://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/sizing.html), after total available memory, the second most influential factor is the proportion of the heap reserved for the Young Generation. By default, the minimum size of the YG is 1310 *MB*, and maximum size is *unlimited*.

We can assign them explicitly:

```
-XX:NewSize=<young size>[unit]
-XX:MaxNewSize=<young size>[unit]
```

3. Garbage Collection

For better stability of the application, choosing of right Garbage Collection (http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html) algorithm is critical.

JVM has four types of *GC* implementations:

- Serial Garbage Collector
- Parallel Garbage Collector
- CMS Garbage Collector
- G1 Garbage Collector

These implementations can be declared with the below parameters:

```
-XX:+UseSerialGC
-XX:+UseParallelGC
-XX:+USeParNewGC
-XX:+UseG1GC
```

More details on *Garbage Collection* implementations can be found here (http://www.baeldung.com/jvm-garbage-collectors).

4. GC Logging

To strictly monitor the application health, we should always check the JVM's *Garbage Collection* performance. The easiest way to do this is to log the *GC* activity in human readable format.

Using the following parameters, we can log the *GC* activity:

```
-XX:+UseGCLogFileRotation
-XX:NumberOfGCLogFiles=< number of log files >
-XX:GCLogFileSize=< file size >[ unit ]
-Xloggc:/path/to/gc.log
```

UseGCLogFileRotation specifies the log file rolling policy, much like log4j, s4lj, etc. ***NumberOfGCLogFiles*** denotes the max number of log files that can be written for a single application life cycle. ***GCLogFileSize*** specifies the max size of the file. Finally, ***loggc*** denotes its location.

Point to note here is that, there are two more JVM parameters available (***-XX:+PrintGCTimeStamps*** and ***-XX:+PrintGCDateStamps***) which can be used to print date-wise timestamp in the *GC* log.

For example, if we want to assign a maximum of 100 GC log files, each having a maximum size of 50 MB and want to store them in '*/home/user/log/*' location, we can use below syntax:

```
-XX:+UseGCLogFileRotation
-XX:NumberOfGCLogFiles=10
-XX:GCLogFileSize=50M
-Xloggc:/home/user/log/gc.log
```

However, the problem is that one additional daemon thread is always used for monitoring system time in the background. This behavior may create some performance bottleneck; that's why it's always better not to play with this parameter in production.

5. Handling Out of Memory

It's very common for a large application to face out of memory error (https://docs.oracle.com/javase/7/docs/api/java/lang/OutOfMemoryError.html) which, in turn, results in the application crash. It's a very critical scenario and very hard to replicate to troubleshoot the issue.

That's why JVM comes with some parameters which dump heap memory into a physical file which can be used later for finding out leaks:

```
-XX:+HeapDumpOnOutOfMemoryError
-XX:HeapDumpPath=./java_pid<pid>.hprof
-XX:OnOutOfMemoryError="< cmd args >;< cmd args >"
-XX:+UseGCOverheadLimit
```

A couple of points to note here:

- ***HeapDumpOnOutOfMemoryError*** instructs the JVM to dump heap into physical file in case of *OutOfMemoryError*
- ***HeapDumpPath*** denotes the path where the file is to be written; any filename can be given; however, if JVM finds a **<pid>** tag in the name, the process id of the current process causing the out of memory error will be appended to the file name with *.hprof* format
- ***OnOutOfMemoryError*** is used to issue emergency commands to be executed in case of out of memory error; proper command should be

used in the space of cmd args. For example, if we want to restart the server as soon as out of memory occur, we can set the parameter:

```
-XX:OnOutOfMemoryError="shutdown -r"
```

- ***UseGCOverheadLimit*** is a policy that limits the proportion of the VM's time that is spent in GC before an *OutOfMemory* error is thrown

6. 32/64 bit

In the OS environment where both 32 and 64-bit packages are installed, the JVM automatically chooses 32-bit environmental packages.

If we want to set the environment to 64 bit manually, we can do so using below parameter:

```
-d<OS bit>
```

OS bit can be either **32** or **64**. More information about this can be found here (http://www.oracle.com/technetwork/java/hotspotfaq-138619.html#64bit_layering).

7. Misc

- ***-server***: enables "Server Hotspot VM"; this parameter is used by default in 64 bit JVM
- ***-XX:+UseStringDeduplication***: *Java 8u20* has introduced this JVM parameter for reducing the unnecessary use of memory by creating too many instances of the same *String;* this optimizes the heap memory by reducing duplicate *String* values to a single global char[] array
- ***-XX:+UseLWPSynchronization***: sets *LWP* (*Light Weight Process*) – based synchronization policy instead of thread-based synchronization
- ***-XX:LargePageSizeInBytes*:** sets the large page size used for the Java heap; it takes the argument in GB/MB/KB; with larger page sizes we can make better use of virtual memory hardware resources; however, this

may cause larger space sizes for the *PermGen*, which in turn can force to reduce the size of Java heap space

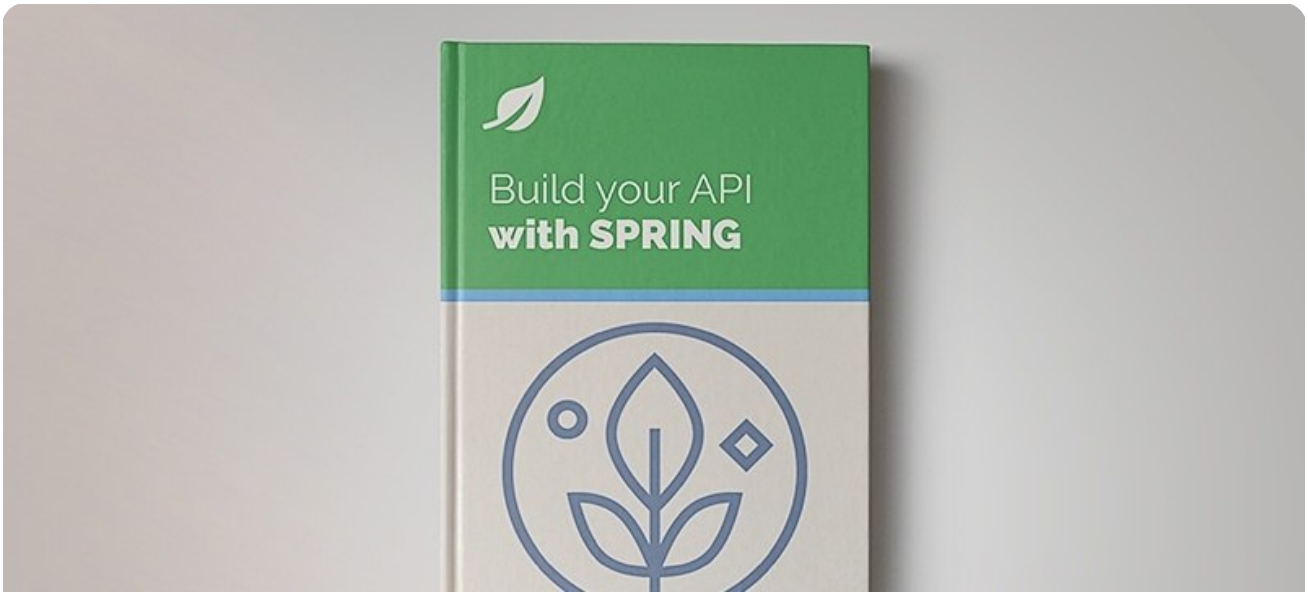
- ***-XX:MaxHeapFreeRatio***: sets the maximum percentage of heap free after *GC* to avoid shrinking.
- ***-XX:MinHeapFreeRatio***: sets the minimum percentage of heap free after *GC* to avoid expansion; to monitor the heap usage you can use **VisualVM (https://visualvm.github.io/)** shipped with JDK.
- ***-XX:SurvivorRatio***: Ratio of *eden/ survivor space size* – for example, *-XX:SurvivorRatio=6* sets the ratio between each *survivor space* and *eden space* to be 1:6,
- ***-XX:+UseLargePages***: use large page memory if it is supported by the system; please note that *OpenJDK 7* tends to crash if using this JVM parameter
- ***-XX:+UseStringCache*:** enables caching of commonly allocated strings available in the *String* pool
- ***-XX:+UseCompressedStrings***: use a *byte[]* type for *String* objects which can be represented in pure ASCII format
- ***-XX:+OptimizeStringConcat*:** it optimizes *String* concatenation operations where possible

8. Conclusion

In this quick article, we learned about some important JVM parameters – which can be used to tune and improve general application performance.

Some of these can also be used for debugging purposes.

If you want to explore the reference parameters in more detail, you can get started here (http://www.oracle.com/technetwork/articles/java/vmoptions-jsp-140102.html).



(http://www.baeldung.com/wp-content/uploads/2016/05/baeldung-rest-post-footer-main-1.2.0.jpg)

(http://www.baeldung.com/wp-content/uploads/2016/05/baeldung-rest-post-footer-icn-1.0.0.png)

Learning to "Build your API **with Spring**"?

Enter your Email Address

Subscribe ▾ ▲ newest ▲ oldest ▲ most voted

Guest

Abhishek Gupta

Thx, Nice overview of the JVM parameters.. it triggered me to look more deeper in each area w.r.t performance .

+ 0 −

1 year ago

Guest

Jitendra Khimji Arethiya

For 5th Point, we have to use -XX: + HeapDumpOnOutOfMemoryError

+ 0 −

1 year ago ^

Guest

Grzegorz Piwowarek

Thanks, good catch!

+ 0 −

1 year ago

CATEGORIES

SPRING (HTTP://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
REST (HTTP://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTP://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SECURITY (HTTP://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCE (HTTP://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)

JACKSON (HTTP://WWW.BAELDUNG.COM/CATEGORY/JACKSON/)
HTTPCLIENT (HTTP://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTP://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIES

JAVA "BACK TO BASICS" TUTORIAL (HTTP://WWW.BAELDUNG.COM/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (HTTP://WWW.BAELDUNG.COM/JACKSON)
HTTPCLIENT 4 TUTORIAL (HTTP://WWW.BAELDUNG.COM/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (HTTP://WWW.BAELDUNG.COM/REST-WITH-SPRING-SERIES/)
SPRING PERSISTENCE TUTORIAL (HTTP://WWW.BAELDUNG.COM/PERSISTENCE-WITH-SPRING-SERIES/)
SECURITY WITH SPRING (HTTP://WWW.BAELDUNG.COM/SECURITY-SPRING)

ABOUT

ABOUT BAELDUNG (HTTP://WWW.BAELDUNG.COM/ABOUT/)
THE COURSES (HTTP://COURSES.BAELDUNG.COM)
CONSULTING WORK (HTTP://WWW.BAELDUNG.COM/CONSULTING)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
THE FULL ARCHIVE (HTTP://WWW.BAELDUNG.COM/FULL_ARCHIVE)
WRITE FOR BAELDUNG (HTTP://WWW.BAELDUNG.COM/CONTRIBUTION-GUIDELINES)
CONTACT (HTTP://WWW.BAELDUNG.COM/CONTACT)
COMPANY INFO (HTTP://WWW.BAELDUNG.COM/BAELDUNG-COMPANY-INFO)
TERMS OF SERVICE (HTTP://WWW.BAELDUNG.COM/TERMS-OF-SERVICE)
PRIVACY POLICY (HTTP://WWW.BAELDUNG.COM/PRIVACY-POLICY)
EDITORS (HTTP://WWW.BAELDUNG.COM/EDITORS)
MEDIA KIT (PDF) (HTTPS://S3.AMAZONAWS.COM/BAELDUNG.COM/BAELDUNG+-+MEDIA+KIT.PDF)