(http://baeldung.com)

JVM Garbage Collectors

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 will show the basics of different *JVM Garbage Collection (GC)* implementations. Additionally, we'll find out how to enable a particular type of Garbage Collection in our applications.

2. Brief Introduction to Garbage Collection

From the name, it looks like *Garbage Collection* deals with finding and deleting the garbage from memory. However, in reality, *Garbage Collection* tracks each and every object available in the JVM heap space and removes unused ones.

In simple words, *GC* works in two simple steps known as Mark and Sweep:

- **Mark –** it is where the garbage collector identifies which pieces of memory are in use and which are not
- **Sweep –** this step removes objects identified during the "mark" phase

Advantages:

- No manual memory allocation/deallocation handling because unused memory space is automatically handled by *GC*
- No overhead of handling ***Dangling Pointer (https://en.wikipedia.org/wiki/Dangling_pointer)***
- Automatic ***Memory Leak (https://en.wikipedia.org/wiki/Memory_leak)*** management (*GC* on its own can't guarantee the full proof solution to memory leaking, however, it takes care of a good portion of it)

Disadvantages:

- Since *JVM* has to keep track of object reference creation/deletion, this activity requires more CPU power besides the original application. It may affect the performance of requests which required large memory
- Programmers have no control over the scheduling of CPU time dedicated to freeing objects that are no longer needed
- Using some GC implementations might result in application stopping unpredictably
- Automatized memory management will not be as efficient as the proper manual memory allocation/deallocation

3. GC Implementations

JVM has four types of *GC* implementations:

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

3.1. Serial Garbage Collector

This is the simplest GC implementation, as it basically works with a single thread. As a result, **this *GC* implementation freezes all application threads when it runs**. Hence, it not a good idea to use it in multi-threaded applications like server environments.

However, there was an excellent talk (https://www.infoq.com/presentations/JVM-Performance-Tuning-twitter-QCon-London-2012) by *Twitter* engineers at QCon 2012 on the performance of *Serial Garbage Collector* – which is a good way to understand this collector better.

The Serial GC is the garbage collector of choice for most applications that do not have small pause time requirements and run on client-style machines. To enable *Serial Garbage Collector*, we can use the following argument:

```
1 java -XX:+UseSerialGC -jar Application.java
```

3.2. Parallel Garbage Collector

It's the default *GC* of the *JVM* and sometimes called Throughput Collectors. Unlike *Serial Garbage Collector*, this **uses multiple threads for managing heap space**. But it also freezes other application threads while performing *GC*.

If we use this *GC*, we can specify maximum garbage collection *threads and pause time, throughput and footprint* (heap size).

The numbers of garbage collector threads can be controlled with the command-line option *-XX:ParallelGCThreads=<N>*.

The maximum pause time goal (gap [in milliseconds] between two *GC*)is specified with the command-line option *-XX:MaxGCPauseMillis=<N>*.

The maximum throughput target (measured regarding the time spent doing garbage collection versus the time spent outside of garbage collection) is specified by the command-line option *-XX:GCTimeRatio=<N>.*

Maximum heap footprint (the amount of heap memory that a program requires while running) is specified using the option *-Xmx<N>.*

To enable *Parallel Garbage Collector*, we can use the following argument:

```
java -XX:+UseParallelGC -jar Application.java
```

3.3. CMS Garbage Collector

The *Concurrent Mark Sweep (CMS)* implementation uses multiple garbage collector threads for garbage collection. It's designed for applications that prefer shorter garbage collection pauses, and that can afford to share processor resources with the garbage collector while the application is running.

Simply put, applications using this type of GC respond slower on average but do not stop responding to perform garbage collection.

A quick point to note here is that since this *GC* is concurrent, an invocation of explicit garbage collection such as using *System.gc()* while the concurrent process is working, will result in *Concurrent Mode Failure / Interruption* (https://blogs.oracle.com/jonthecollector/entry/what_the_heck_s_a).

If more than 98% of the total time is spent in *CMS* garbage collection and less than 2% of the heap is recovered, then an *OutOfMemoryError* is thrown by the *CMS collector*. If necessary, this feature can be disabled by adding the option *-XX:-UseGCOverheadLimit* to the command line.

This collector also has a mode knows as an incremental mode which is being deprecated in Java SE 8 and may be removed in a future major release.

To enable the *CMS Garbage Collector*, we can use the following flag:

```
java -XX:+USeParNewGC -jar Application.java
```

3.4. G1 Garbage Collector

G1 (Garbage First) Garbage Collector is designed for applications running on multi-processor machines with large memory space. It's available since *JDK7 Update 4* and in later releases.

G1 collector will replace the *CMS* collector since it's more performance efficient.

Unlike other collectors, *G1* collector partitions the heap into a set of equal-sized heap regions, each a contiguous range of virtual memory. When performing garbage collections, *G1* shows a concurrent global marking phase

(i.e. phase 1 known as *Marking)* to determine the liveness of objects throughout the heap.

After the mark phase is completed, *G1* knows which regions are mostly empty. It collects in these areas first, which usually yields a significant amount of free space (i.e. phase 2 known as *Sweeping).* It is why this method of garbage collection is called Garbage-First.

To enable *G1 Garbage Collector*, we can use the following argument:

```
1 java -XX:+UseG1GC -jar Application.java
```

3.5. Java 8 Changes

Java 8u20 has introduced one more *JVM* parameter for reducing the unnecessary use of memory by creating too many instances of same *String.* This optimizes the heap memory by removing duplicate *String* values to a global single *char[]* array.

This parameter can be enabled by adding ***-XX:+UseStringDeduplication*** as *JVM* parameter.

4. Conclusion

In this quick tutorial, we had a look at the different *JVM Garbage Collection* implementations and their use cases.

More detailed documentation can be found here (http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.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

>> Get the eBook

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)