Garbage Collection -2024

Java offers many garbage collectors to meet different application requirements.

**Why GC**? GC addresses three major factors.

**Throughput**:. Throughput is time available for the application threads (non-gc threads) to do their tasks seamlessly without getting blocked by the gc collections. For example, if your throughput is 95%, that means the application code is running 95% of the time and garbage collection is running 5% of the time.

**Latency:**. Latency is the responsiveness of an application. Garbage collection pauses affect the responsiveness of applications.

**Footprint**:. *Footprint is the amount of memory allocated by a process.* ***Footprint is the working size of the JVM process, measured in pages and cache lines****.* ***Promptness is the time between when an object becomes dead, and when the memory becomes available****.*

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

JVM has following types of *GC* implementations:

* Serial Garbage Collector
* Parallel Garbage Collector
* CMS Garbage Collector (Deprecated)
* G1 Garbage Collector
* Z Garbage Collector
* Shenandoah collector

# **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 is not a good idea to use it in multi-threaded applications like server environments**. 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.

**Usage: java -XX:+UseSerialGC -jar Application.java**

# **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). it is often the best choice when throughput is more important than latency.

**Usage: java -XX:+UseParallelGC -jar Application.java**

# **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. [**As of Java 9**](https://openjdk.java.net/jeps/291)**, the CMS garbage collector has been deprecated** with the recommendation to use the G1 collector instead.

**Usage: java -XX:+UseParNewGC -jar Application.java**

# **G1 Garbage Collector**

**G1 GC (Garbage First Garbage Collector) is a low-pause garbage collector that divides the heap into regions and performs garbage collection on a per-region basis. The goal is to minimize the pause time for individual regions by performing concurrent garbage collection in the background. G1 (Garbage First) Garbage Collector is designed for applications running on multi-processor machines with large memory space.** Unlike other collectors, the ***G1* collector divides the heap regions. When performing garbage collections, *G1* performs 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.

**Usage: java -XX:+UseG1GC -jar Application.java**

# **Java 8 Changes**

**Java 8u20 has introduced one more JVM parameter for reducing the unnecessary use of memory by creating too many instances of the same String.**

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

# **Z Garbage Collector**

[ZGC (Z Garbage Collector)](https://www.baeldung.com/jvm-zgc-garbage-collector)is a scalable **low-latency garbage collector** which debuted in Java 11. ZGC performs all expensive work concurrently,**without stopping the execution of application threads for more than 10 ms**, which makes it suitable for applications that require low latency. It uses**load barriers with colored pointers** to perform concurrent operations when the threads are running and they are used to keep track of heap usage. Similar to G1, Z Garbage Collector partitions the heap, except that heap regions can have different sizes. This collector is suitable for applications with very large amounts of memory that require very short pause times.

To enable the Z Garbage Collector, we can use the following argument in JDK versions lower than 15:

**Usage: java -XX:+UnlockExperimentalVMOptions -XX:+UseZGC Application.java**

From version 15 we don't need experimental mode on:

**Usage: java -XX:+UseZGC Application.java**

# **Shenandoah Garbage Collector**

Shenandoah is another garbage collector with very short pause times. It reduces pause times by performing more garbage collection work concurrently with the application, including concurrent compaction. Shenandoah's pause time is independent of the heap size. Garbage collecting a 2GB heap or a 200GB heap should have a similar short pause behavior. Shenandoah is best suited to an application that needs responsiveness and short pause times, irrespective of heap size requirements.

**Usage: java -XX:+** **UseShenandoahGC Application.java**

**Note**: In JDK 11 and 17 Serial collector is used when there is only one CPU available. Otherwise G1 is selected

### **Selecting a Collector – A starting guidelines**

* If the application has a small data set (up to approximately 100 MB), then select the serial collector with the option -XX:+UseSerialGC.
* For the following condition, then let the VM select the collector or select the parallel collector with -XX:+UseParallelGC.
  + application performance is the first priority
  + there are no pause-time
* If response time is more important than overall throughput and garbage collection pauses must be kept shorter than approximately one second, then select a mostly concurrent collector with -XX:+UseG1GC or -XX:+UseConcMarkSweepGC.
* If response time is a high priority, and/or you are using a very large heap, then select a fully concurrent collector with -XX:UseZGC.

most production engine servers

-Xms24G -Xmx24G -XX:PermSize=512m -XX:+UseG1GC -XX:MaxGCPauseMillis=200 -XX:ParallelGCThreads=20 -XX:ConcGCThreads=5 -XX:InitiatingHeapOccupancyPercent=70

For standalone installations, use the example settings:

-Xms32G -Xmx32G -XX:PermSize=512m -XX:+UseG1GC -XX:MaxGCPauseMillis=200 -XX:ParallelGCThreads=20 -XX:ConcGCThreads=5 -XX:InitiatingHeapOccupancyPercent=70

* **-Xms, -Xmx**: Places boundaries on the heap size to increase the predictability of garbage collection. The heap size is limited in replica servers so that even Full GCs do not trigger SIP retransmissions. **-Xms** sets the starting size to prevent pauses caused by heap expansion.
* **-XX:+UseG1GC**: Use the Garbage First (G1) Collector.
* **-XX:MaxGCPauseMillis**: Sets a target for the maximum GC pause time. This is a soft goal, and the JVM will make its best effort to achieve it.
* **-XX:ParallelGCThreads**: Sets the number of threads used during parallel phases of the garbage collectors. The default value varies with the platform on which the JVM is running.
* **-XX:ConcGCThreads**: Number of threads concurrent garbage collectors will use. The default value varies with the platform on which the JVM is running.
* **-XX:InitiatingHeapOccupancyPercent**: Percentage of the (entire) heap occupancy to start a concurrent GC cycle. GCs that trigger a concurrent GC cycle based on the occupancy of the entire heap and not just one of the generations, including G1, use this option. A value of 0 denotes 'do constant GC cycles'. The default value is 45.