Garbage Collection -2022

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 five types of *GC* implementations:

* Serial Garbage Collector
* Parallel Garbage Collector
* CMS Garbage Collector
* G1 Garbage Collector
* Z Garbage 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**.

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

### **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. Unlike other collectors, the ***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.

**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.**

his parameter 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 as an experimental option for Linux. JDK 14 introduced  ZGC under the Windows and macOS operating systems. ZGC has obtained the production status from Java 15 onwards.

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.

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**