-XX:MaxPermSize - size of the Permanent Generation (ex: -XX:MaxPermSize=30m );

-Xmx : max heap size (ex: -Xmx1024);

-Xms : min heap size (initial);

-Xmn : the size of the heap for the young generation.

0) Default configuration:

java -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -jar apptorun.jar

Output:

INFO - GC: PS Scavenge - java.lang:type=GarbageCollector,name=PS Scavenge

INFO - GC: PS MarkSweep - java.lang:type=GarbageCollector,name=PS MarkSweep

Heap

PSYoungGen total 38400K, used 2671K [0x00000007d5e00000, 0x00000007d8880000, 0x0000000800000000)

eden space 33280K, 8% used [0x00000007d5e00000,0x00000007d609bf10,0x00000007d7e80000)

from space 5120K, 0% used [0x00000007d8380000,0x00000007d8380000,0x00000007d8880000)

to space 5120K, 0% used [0x00000007d7e80000,0x00000007d7e80000,0x00000007d8380000)

ParOldGen total 86016K, used 0K [0x0000000781a00000, 0x0000000786e00000, 0x00000007d5e00000)

object space 86016K, 0% used [0x0000000781a00000,0x0000000781a00000,0x0000000786e00000)

PSPermGen total 21504K, used 4405K [0x000000077c800000, 0x000000077dd00000, 0x0000000781a00000)

object space 21504K, 20% used [0x000000077c800000,0x000000077cc4d4d8,0x000000077dd00000)

1) Task 1

Write a Command Line that starts a JavaApp using the Serial Collector with the following parameters

* the 6m initial heap size for when the JVM starts
* the18m maximum heap size
* the2m size of the Young Generation
* the 20m starting size of the Permanent Generation
* the 30 maximum size of the Permanent Generation

java -XX:+UseSerialGC -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -Xms6m -Xmx18m -Xmn2m -XX:PermSize=20m -XX:MaxPermSize=30m -jar apptorun.jar

Output:

0.122: [GC0.122: [DefNew: 1664K->191K(1856K), 0.0148725 secs] 1664K->647K(5952K), 0.0155604 secs] [Times: user=0.00 sys=0.00, real=0.02 secs]

INFO - GC: Copy - java.lang:type=GarbageCollector,name=Copy

INFO - GC: MarkSweepCompact - java.lang:type=GarbageCollector,name=MarkSweepCompact

Heap

def new generation total 1856K, used 451K [0x00000000fd000000, 0x00000000fd200000, 0x00000000fd200000)

eden space 1664K, 15% used [0x00000000fd000000, 0x00000000fd040d88, 0x00000000fd1a0000)

from space 192K, 99% used [0x00000000fd1d0000, 0x00000000fd1ffff8, 0x00000000fd200000)

to space 192K, 0% used [0x00000000fd1a0000, 0x00000000fd1a0000, 0x00000000fd1d0000)

tenured generation total 4096K, used 455K [0x00000000fd200000, 0x00000000fd600000, 0x00000000fe200000)

the space 4096K, 11% used [0x00000000fd200000, 0x00000000fd271c60, 0x00000000fd271e00, 0x00000000fd600000)

compacting perm gen total 20480K, used 4405K [0x00000000fe200000, 0x00000000ff600000, 0x0000000100000000)

the space 20480K, 21% used [0x00000000fe200000, 0x00000000fe64d4d8, 0x00000000fe64d600, 0x00000000ff600000)

No shared spaces configured.

.

2) Task 2

Write a Command Line that starts a JavaApp using the Parallel Collector with the following parameters

* the 3m initial heap size for when the JVM starts
* the12m maximum heap size
* the1m size of the Young Generation
* the 20m starting size of the Permanent Generation
* the 20 maximum size of the Permanent Generation

java -XX:+UseParallelGC -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -Xms3m -Xmx12m -Xmn1m -XX:PermSize=20m -XX:MaxPermSize=20m -jar apptorun.jar

Output:

INFO - GC: PS Scavenge - java.lang:type=GarbageCollector,name=PS Scavenge

INFO - GC: PS MarkSweep - java.lang:type=GarbageCollector,name=PS MarkSweep

Heap

PSYoungGen total 512K, used 0K [0x00000000fff00000, 0x0000000100000000, 0x0000000100000000)

eden space 0K, -2147483648% used [0x00000000fff00000,0x00000000fff00000,0x00000000fff00000)

from space 512K, 0% used [0x00000000fff80000,0x00000000fff80000,0x0000000100000000)

to space 512K, 0% used [0x00000000fff00000,0x00000000fff00000,0x00000000fff80000)

ParOldGen total 2048K, used 1877K [0x00000000ff400000, 0x00000000ff600000, 0x00000000fff00000)

object space 2048K, 91% used [0x00000000ff400000,0x00000000ff5d5640,0x00000000ff600000)

PSPermGen total 20480K, used 4405K [0x00000000fe000000, 0x00000000ff400000, 0x00000000ff400000)

object space 20480K, 21% used [0x00000000fe000000,0x00000000fe44d4d8,0x00000000ff400000)

3) Task 3

Write a Command Line that starts a JavaApp using the Parallel Old Collector with the following parameters

* the 9m initial heap size for when the JVM starts
* the18m maximum heap size
* the3m size of the Young Generation
* the 40m starting size of the Permanent Generation
* the 40 maximum size of the Permanent Generation

java -XX:+UseParallelOldGC -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -Xms9m -Xmx18m -Xmn3m -XX:PermSize=40m -XX:MaxPermSize=40m -jar apptorun.jar

Output:

INFO - GC: PS Scavenge - java.lang:type=GarbageCollector,name=PS Scavenge

INFO - GC: PS MarkSweep - java.lang:type=GarbageCollector,name=PS MarkSweep

Heap

PSYoungGen total 2560K, used 1963K [0x00000000ffd00000, 0x0000000100000000, 0x0000000100000000)

eden space 2048K, 95% used [0x00000000ffd00000,0x00000000ffeeade8,0x00000000fff00000)

from space 512K, 0% used [0x00000000fff80000,0x00000000fff80000,0x0000000100000000)

to space 512K, 0% used [0x00000000fff00000,0x00000000fff00000,0x00000000fff80000)

ParOldGen total 6144K, used 0K [0x00000000fee00000, 0x00000000ff400000, 0x00000000ffd00000)

object space 6144K, 0% used [0x00000000fee00000,0x00000000fee00000,0x00000000ff400000)

PSPermGen total 40960K, used 4405K [0x00000000fc600000, 0x00000000fee00000, 0x00000000fee00000)

object space 40960K, 10% used [0x00000000fc600000,0x00000000fca4d4d8,0x00000000fee00000)

4) Task 4

Write a Command Line that starts a JavaApp using the Concurrent Mark Sweep (CMS) Collector with the following parameters

* the 6m initial heap size for when the JVM starts
* the18m maximum heap size
* the 2m size of the Young Generation
* the 20m starting size of the Permanent Generation
* the 30 maximum size of the Permanent Generatio

java -XX:+UseConcMarkSweepGC -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -Xms6m -Xmx18m -Xmn2m -XX:PermSize=20m -XX:MaxPermSize=30m -jar apptorun.jar

Output:

0.118: [GC0.119: [ParNew: 1657K->192K(1856K), 0.0025116 secs] 1657K->648K(5952K), 0.0032545 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]

INFO - GC: ParNew - java.lang:type=GarbageCollector,name=ParNew

INFO - GC: ConcurrentMarkSweep - java.lang:type=GarbageCollector,name=ConcurrentMarkSweep

Heap

par new generation total 1856K, used 511K [0x00000000fd000000, 0x00000000fd200000, 0x00000000fd200000)

eden space 1664K, 19% used [0x00000000fd000000, 0x00000000fd04fdf0, 0x00000000fd1a0000)

from space 192K, 100% used [0x00000000fd1d0000, 0x00000000fd200000, 0x00000000fd200000)

to space 192K, 0% used [0x00000000fd1a0000, 0x00000000fd1a0000, 0x00000000fd1d0000)

concurrent mark-sweep generation total 4096K, used 456K [0x00000000fd200000, 0x00000000fd600000, 0x00000000fe200000)

concurrent-mark-sweep perm gen total 20480K, used 4406K [0x00000000fe200000, 0x00000000ff600000, 0x0000000100000000)

5) Task 5

Write a Command Line that starts a JavaApp using the Concurrent Mark Sweep (CMS) Collector with 2 Parallel CMS Threads with the following parameters

* the 2m initial heap size for when the JVM starts
* the18m maximum heap size
* the 1m size of the Young Generation
* the 24m starting size of the Permanent Generation
* the 36 maximum size of the Permanent Generation

java -XX:+UseConcMarkSweepGC -XX:ParallelCMSThreads=2 -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -Xms2m -Xmx18m -Xmn1m -XX:PermSize=24m -XX:MaxPermSize=36m -jar apptorun.jar

Output:

0.074: [GC0.074: [ParNew: 890K->64K(960K), 0.0017857 secs] 890K->419K(1984K), 0.0021940 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]

0.122: [GC0.122: [ParNew: 960K->63K(960K), 0.0118926 secs] 1315K->698K(2368K), 0.0124626 secs] [Times: user=0.00 sys=0.00, real=0.01 secs]

INFO - GC: ParNew - java.lang:type=GarbageCollector,name=ParNew

INFO - GC: ConcurrentMarkSweep -java.lang:type=GarbageCollector,name=ConcurrentMarkSweep

Heap

par new generation total 960K, used 186K [0x00000000fca00000, 0x00000000fcb00000, 0x00000000fcb00000)

eden space 896K, 13% used [0x00000000fca00000, 0x00000000fca1e9d0, 0x00000000fcae0000)

from space 64K, 99% used [0x00000000fcae0000, 0x00000000fcaefec0, 0x00000000fcaf0000)

to space 64K, 0% used [0x00000000fcaf0000, 0x00000000fcaf0000, 0x00000000fcb00000)

concurrent mark-sweep generation total 1408K, used 635K [0x00000000fcb00000, 0x00000000fcc60000, 0x00000000fdc00000)

concurrent-mark-sweep perm gen total 24576K, used 4406K [0x00000000fdc00000, 0x00000000ff400000, 0x0000000100000000)

6) Task 6

Write a Command Line that starts a JavaApp using the Parallel Collector with 2 Parallel CMS Threads with the following parameters

* the 4m initial heap size for when the JVM starts
* the16m maximum heap size
* the3m size of the Young Generation
* the 24m starting size of the Permanent Generation
* the 32 maximum size of the Permanent Generation

java -XX:+UseParNewGC -XX:ParallelCMSThreads=2 -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -Xms4m -Xmx16m -Xmn3m -XX:PermSize=24m -XX:MaxPermSize=32m -jar apptorun.jar

Output:

INFO - GC: ParNew - java.lang:type=GarbageCollector,name=ParNew

INFO - GC: MarkSweepCompact - java.lang:type=GarbageCollector,name=MarkSweepCompact

Heap

par new generation total 2816K, used 1952K [0x00000000fd000000, 0x00000000fd300000, 0x00000000fd300000)

eden space 2560K, 76% used [0x00000000fd000000, 0x00000000fd1e8280, 0x00000000fd280000)

from space 256K, 0% used [0x00000000fd280000, 0x00000000fd280000, 0x00000000fd2c0000)

to space 256K, 0% used [0x00000000fd2c0000, 0x00000000fd2c0000, 0x00000000fd300000)

tenured generation total 1024K, used 0K [0x00000000fd300000, 0x00000000fd400000, 0x00000000fe000000)

the space 1024K, 0% used [0x00000000fd300000, 0x00000000fd300000, 0x00000000fd300200, 0x00000000fd400000)

compacting perm gen total 24576K, used 4405K [0x00000000fe000000, 0x00000000ff800000, 0x0000000100000000)

the space 24576K, 17% used [0x00000000fe000000, 0x00000000fe44d4d8, 0x00000000fe44d600, 0x00000000ff800000)

No shared spaces configured.

7) Task 7

Write a Command Line that starts a JavaApp using theG1 Garbage Collector with the following parameters

* the 4m initial heap size for when the JVM starts
* the16m maximum heap size
* them 2m size of the Young Generation
* the 12m starting size of the Permanent Generation
* the 18 maximum size of the Permanent Generation

java -XX:+UseG1GC -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -Xms4m -Xmx16m -Xmn2m -XX:PermSize=12m -XX:MaxPermSize=18m -jar apptorun.jar

Output:

INFO - GC: G1 Young Generation - java.lang:type=GarbageCollector,name=G1 Young Generation

INFO - GC: G1 Old Generation - java.lang:type=GarbageCollector,name=G1 Old Generation

Heap

garbage-first heap total 4096K, used 1024K [0x00000000fde00000, 0x00000000fe200000, 0x00000000fee00000)

region size 1024K, 2 young (2048K), 0 survivors (0K)

compacting perm gen total 12288K, used 4405K [0x00000000fee00000, 0x00000000ffa00000, 0x0000000100000000)

the space 12288K, 35% used [0x00000000fee00000, 0x00000000ff24d4d8, 0x00000000ff24d600, 0x00000000ffa00000)

No shared spaces configured.

**Young generation collectors**

**Copy (enabled with -XX:+UseSerialGC)** -

the serial copy collector, uses one thread to copy surviving objects from Eden to Survivor spaces and between Survivor spaces until it decides they've been there long enough, at which point it copies them into the old generation.

**PS Scavenge (enabled with -XX:+UseParallelGC)** -

the parallel scavenge collector, like the **Copy** collector, but uses multiple threads in parallel and has some knowledge of how the old generation is collected (essentially written to work with the serial and PS old gen collectors).

**ParNew (enabled with -XX:+UseParNewGC)** -

the parallel copy collector, like the **Copy** collector, but uses multiple threads in parallel and has an internal 'callback' that allows an old generation collector to operate on the objects it collects (really written to work with the concurrent collector).

**G1 Young Generation (enabled with -XX:+UseG1GC)** -

the garbage first collector, uses the 'Garbage First' algorithm which splits up the heap into lots of smaller spaces, but these are still separated into Eden and Survivor spaces in the young generation for G1.

**Old generation collectors**

**MarkSweepCompact (enabled with -XX:+UseSerialGC)** -

the serial mark-sweep collector, the daddy of them all, uses a serial (one thread) full mark-sweep garbage collection algorithm, with optional compaction.

**PS MarkSweep (enabled with -XX:+UseParallelOldGC)** -

the parallel scavenge mark-sweep collector, parallelised version (i.e. uses multiple threads) of the **MarkSweepCompact**.

**ConcurrentMarkSweep (enabled with -XX:+UseConcMarkSweepGC)** -

the concurrent collector, a garbage collection algorithm that attempts to do most of the garbage collection work in the background without stopping application threads while it works (there are still phases where it has to stop application threads, but these phases are attempted to be kept to a minimum). Note if the concurrent collector fails to keep up with the garbage, it fails over to the serial **MarkSweepCompact** collector for (just) the next GC.

**G1 Mixed Generation (enabled with -XX:+UseG1GC)** -

the garbage first collector, uses the 'Garbage First' algorithm which splits up the heap into lots of smaller spaces.

All of the garbage collection algorithms except **ConcurrentMarkSweep** are stop-the-world, i.e. they stop all application threads while they operate - the stop is known as 'pause' time. The **ConcurrentMarkSweep** tries to do most of it's work in the background and minimize the pause time, but it also has a stop-the-world phase and can fail into the **MarkSweepCompact** which is fully stop-the-world.

The full list of possible GC algorithm combinations that can work are:

|  |  |
| --- | --- |
| **Command Options\*** | **Resulting Collector Combination** |
| **-XX:+UseSerialGC** | young **Copy** and old **MarkSweepCompact** |
| **-XX:+UseG1GC** | young **G1 Young** and old **G1 Mixed** |
| **-XX:+UseParallelGC -XX:+UseParallelOldGC -XX:+UseAdaptiveSizePolicy** | young **PS Scavenge** old **PS MarkSweep** with adaptive sizing |
| **-XX:+UseParallelGC -XX:+UseParallelOldGC -XX:-UseAdaptiveSizePolicy** | young **PS Scavenge** old **PS MarkSweep**, no adaptive sizing |
| **-XX:+UseParNewGC** | young **ParNew** old **MarkSweepCompact** |
| **-XX:+UseConcMarkSweepGC -XX:+UseParNewGC** | young **ParNew** old **ConcurrentMarkSweep**\*\* |
| **-XX:+UseConcMarkSweepGC -XX:-UseParNewGC** | young **Copy** old **ConcurrentMarkSweep**\*\* |
| \*All the combinations listed here will fail to let the JVM start if you add another GC algorithm not listed, with the exception of -XX:+UseParNewGC which is only combinable with -XX:+UseConcMarkSweepGC | |
| \*\*there are many many options for use with -XX:+UseConcMarkSweepGC which change the algorithm, e.g.   * -XX:+/-CMSIncrementalMode - uses or disables an incremental concurrent GC algorithm * -XX:+/-CMSConcurrentMTEnabled - uses or disables parallel (multiple threads) concurrent GC algorithm * -XX:+/-UseCMSCompactAtFullCollection - uses or disables a compaction when a full GC occurs | |

Other options equivalent to one of the above:

|  |  |
| --- | --- |
| **Command Options Used On Their Own** | **Equivalent To Entry In Table Above** |
| **-XX:+UseParallelGC** | -XX:+UseParallelGC -XX:+UseParallelOldGC |
| **-XX:+UseParallelOldGC** | -XX:+UseParallelGC -XX:+UseParallelOldGC |
| **-Xincgc** | -XX:+UseParNewGC -XX:+UseConcMarkSweepGC |
| **-XX:+UseConcMarkSweepGC** | -XX:+UseParNewGC -XX:+UseConcMarkSweepGC |
| **no option on most Windows** | -XX:+UseSerialGC (see also [this page](http://www.techpaste.com/2012/02/default-jvm-settings-gc-jit-java-heap-sizes-xms-xmx-operating-systems/#more-3569)) |
| **no option on most Unix** | -XX:+UseParallelGC -XX:+UseParallelOldGC -XX:+UseAdaptiveSizePolicy (see also [this page](http://www.techpaste.com/2012/02/default-jvm-settings-gc-jit-java-heap-sizes-xms-xmx-operating-systems/#more-3569)) |
| **-XX:+AggressiveHeap** | -XX:+UseParallelGC -XX:+UseParallelOldGC -XX:+UseAdaptiveSizePolicy with a bunch of other options related to sizing memory and threads and how they interact with the OS |