[JVM调优必备理论知识](https://www.cnblogs.com/ssskkk/p/13063234.html)

**阅读目录**

[对象从出生到消亡过程](https://www.cnblogs.com/ssskkk/p/13063234.html#_label0)

[常见的垃圾回收器](https://www.cnblogs.com/ssskkk/p/13063234.html#_label1)

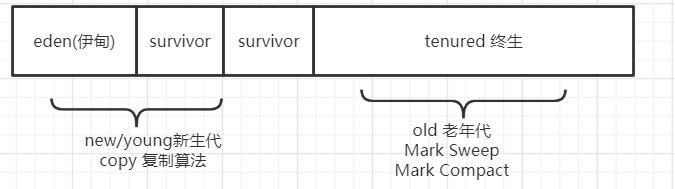
[什么是调优](https://www.cnblogs.com/ssskkk/p/13063234.html#_label2)

[调优步骤](https://www.cnblogs.com/ssskkk/p/13063234.html#_label3)

[G1算法](https://www.cnblogs.com/ssskkk/p/13063234.html#_label4)

[回到顶部](https://www.cnblogs.com/ssskkk/p/13063234.html#_labelTop)

**对象从出生到消亡过程**



新生代概念

新生代分为一个eden区和两个survivor区，默认的比例是8:1:1

eden区是我们new出来对象之后往里面扔的那块区，回收一次跑到survivor

新生代大量死去少量存活 采用复制算法

老年代概念

老年代存活率高回收较少 采用MC或MS

**YGC：年轻代耗尽时触发**

**FullGC：在老年代无法继续分配空间时触发，新生代老年代同时进行回收，所谓JVM调优就是尽量减少FGC**

栈上分配：(栈上分配不需要垃圾回收，直接弹出就没了)

　　-线程私有小对象

　　-无逃逸

　　-支持标量替换

　　-无需调整

线程本地分配(Thread Local Allocation Buffer)：(不需要和其它线程去争用)

　**-占用eden 默认1%(每个线程在eden区取1%空间)**

　　-多线程的时候 不用竞争eden就可以申请空间 提高效率

　　-小对象

　　-无需调整

老年代：

　　**-大对象(大对象直接进入老年代)**

**一个对象从出生到消亡**

**一个对象产生之后首先在栈上分配，栈上分配不下就会进入eden区**

**eden经过一次垃圾回收之后，进入survivor区(s1)，**

**s1在经过一次垃圾回收之后进入s2**

**此后在s1和s2游走，**

**什么时候年龄够了 进入老年代**

[**补充：动态年龄**](https://www.jianshu.com/p/989d3b06a49d)

[回到顶部](https://www.cnblogs.com/ssskkk/p/13063234.html#_labelTop)

**常见的垃圾回收器**

1.JDK诞生 Serial追随 提高效率，诞生了PS，为了配合CMS，诞生了PN，CMS是1.4版本后期引入，CMS是里程碑式的GC，

它开启了并发回收的过程，但是CMS毛病较多，因此目前任何一个JDK版本默认是CMS 并发垃圾回收是因为无法忍受STW

**2.Serial 年轻代 串行回收(a stop-the-word,copying collector which uses a single gc thread) STW时间非常长**

**3.PS Parallel Scavenge  年轻代 并行回收(a stop-the-word copying collector which uses multiple GC threads)**

**4.ParNew 年轻代 配合CMS的并行回收**

**5.SerialOld (a stop-the-word，mark-sweep-compact collector that uses a single GC thread)**

**6.ParallelOld(a compacting collector that uses multiple GC threads)**

**7.ConcurrentMarkSweep** **老年代 并发的(FGC)**， **垃圾回收和应用程序同时运行，降低STW的时间(10g内存十几秒—>200ms)** CMS问题比较多，所以现在没有一个版本默认是CMS，只能手工指定。

**CMS既然是MarkSweep，就一定会有碎片化的问题，碎片到达一定程度，CMS的老年代分配对象分配不下的时候，使用SerialOld 进行老年代回收.**

**解决方案：保持老年代有足够的空间 -XX:CMSInitiatingOccupancyFraction 92%降低这个值：这个值的意思是CMS收集器当老年代使用了92%的时候会被激活。**

**JDK1.5的默认设置下CMS收集器当老年代使用了68%后会被激活，这是一个偏保守的设置。适当的提高这个值可以降低内存回收次数从而获得更好的性能。**

**这个值的意思是CMS收集器当老年代使用了92%的时候会被激活。越大 回收次数越低，但是每次时间越长。越小 回收次数就越多，每次时间就越低**

想象一下： PS + PO -> 加内存 换垃圾回收器 -> PN + CMS + SerialOld（几个小时 - 几天的STW） 几十个G的内存，单线程回收 -> G1 + FGC 几十个G -> 上T内存的服务器 ZGC 算法：三色标记 + Incremental Update

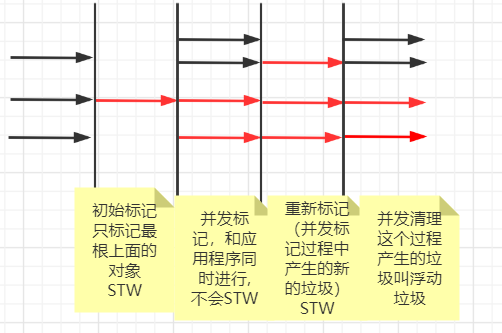
**CMS常见的几个阶段：**

**—>初始标记(只标记最根上的对象STW)**

**—>并发标记(百分之八十时间)**

**—>重新标记(STW 并发标记过程中产生新的垃圾 重新标记)**

**—>并发清理(这个过程产生的垃圾叫浮动垃圾 只能等下一次CMS)**



**8.G1(10ms) 算法：三色标记 + SATB  G1的响应时间高于PN+CMS  但是吞吐量没有PN+CMS高(少百分之十)**

**垃圾收集器跟内存大小的关系**

Serial 几十兆

PS 上百兆 - 几个G

CMS - 20G

G1 - 上百G

ZGC - 4T - 16T（JDK13）

**区分概念：**

内存泄漏memory leak：有一块内存被一个废了的对象占用 也不回收它，叫内存泄露，内存泄露不一定内存溢出

内存溢出out of memory：不断产生对象 ，内存撑不住了

吞吐量：用户代码时间 / (用户代码执行时间+垃圾回收时间) 吞吐量越大说明干正常事的时间越多。

响应时间：STW越短，响应时间越好

所谓调优首先确定追求啥？吞吐量优先还是响应时间优先？在满足一定响应时间的情况下，要求达到多大的吞吐量......

科学计算，数据挖掘，thrput 。吞吐量优先的一般：(PS+PO)

**响应时间优先：网站 GUI API(1.8 G1 也可以ParNew+CMS)**

[回到顶部](https://www.cnblogs.com/ssskkk/p/13063234.html#_labelTop)

**什么是调优**

1.根据需求进行JVM的规划和预调优

2.优化运行JVM运行环境

3.解决JVM运行过程中出现的各种问题(OOM)

[回到顶部](https://www.cnblogs.com/ssskkk/p/13063234.html#_labelTop)

**调优步骤**

2. java -Xms200M -Xmx200M -XX:+PrintGC com.mashibing.jvm.gc.T15\_FullGC\_Problem01

3. 一般是运维团队首先受到报警信息（CPU Memory）

4. top命令观察到问题：内存不断增长 CPU占用率居高不下

5.top -Hp pid可以查看某个进程的**线程**信息，-H 显示线程信息，-p指定pid

6. jstack 定位**线程**状况，jstack threadId > test.txt**(linux命令行显示的线程id是十进制 dump下来的线程id是十六进制)**

　　重点关注：WAITING BLOCKED

eg. waiting on <0x0000000088ca3310> (a java.lang.Object) 假如有一个进程中100个线程，很多线程都在waiting on ，**一定要找到是哪个线程持有这把锁 怎么找？**搜索jstack dump的信息，找 ，看哪个线程持有这把锁RUNNABLE

7. 为什么阿里规范里规定，线程的名称（尤其是线程池）都要写有意义的名称 怎么样自定义线程池里的线程名称？（自定义ThreadFactory）

8. jinfo pid(把该进程详细的虚拟机信息列出来 启动参数等 )

9. jstat -gc 动态观察gc情况 / 阅读GC日志发现频繁GC / arthas观察 / jconsole/jvisualVM/ Jprofiler（最好用） jstat -gc 4655 500 : 每个500个毫秒打印GC的情况 如果面试官问你是怎么定位OOM问题的？如果你回答用图形界面（错误）

　　1：已经上线的系统不用图形界面用什么？（cmdline arthas）

　　2：图形界面到底用在什么地方？测试！测试的时候进行监控！（压测观察）

10. jmap - histo 4655 | head -20，查找有多少对象产生

11. jmap -dump:format=b,file=xxx pid ：

　　线上系统，内存特别大，jmap执行期间会对进程产生很大影响，甚至卡顿（电商不适合）

　　1：设定了参数HeapDump，OOM的时候会自动产生堆转储文件

　　2：很多服务器备份（高可用），停掉这台服务器对其他服务器不影响

　　3：在线定位(一般小点儿公司用不到)

12. java -Xms20M -Xmx20M -XX:+UseParallelGC -XX:+HeapDumpOnOutOfMemoryError com.mashibing.jvm.gc.T15\_FullGC\_Problem01

13. 使用MAT / jhat /jvisualvm 进行dump文件分析 https://www.cnblogs.com/baihuitestsoftware/articles/6406271.html jhat -J-mx512M xxx.dump http://192.168.17.11:7000 拉到最后：找到对应链接 可以使用OQL查找特定问题对象

找到代码的问题

[回到顶部](https://www.cnblogs.com/ssskkk/p/13063234.html#_labelTop)

**G1算法**

以前在物理上是两块连续的内存，一块年轻代 一块老年代，空间太大，无论怎么样也提升不了多少了。

然后利用了分而治之的思想，现在数据量越来越大，分而治之的思想越来越重要

分而治之和分层是架构设计的两大思想。

G1把内存分层了一个个的Region (1 2 4...32)

每个Region在逻辑上依然属于某一个分代，old区 survioe区 Eden区 Humongous区(大对象区)

**每次回收 垃圾最多的那些个Region 也就是G1的名字由来(garbage first)，**

**因为G1产生FGC 和CMS一样也是用的serialOld(JDK10以后会用并行的)**

**G1和CMS调优目标一样，尽量别有FGC**

**可以降低降低MixedGC触发值，让MixedGC提早发生(FGC的频率会变低)**

**XX:InitiatingHeapOccupancyPercent - 启动并发GC时的堆内存占用百分比.**

**默认值45%**

**MixedGC相当于一个完整的CMS**

G1的特点

　　并发收集

　　压缩空闲空间不会延长GC的暂停时间

　　更容易预测GC暂停时间

　　　　G1还有一个特点，新生代和老年代的比例是动态的，不需要手动指定，因为这是G1预测停顿时间的基准5%-60%

　　　　如果STW的时间比较长，G1就会把Y区稍微调小一点，不用机器停掉 手工去调

控制可预测GC的暂停时间。

　　适用不需要实现很高吞吐量的场景，但需要很高响应时间

转载GC调优注意点：

G1调优注意点

Mixed GC 期间老年代空间被填满（晋升失败）

类似 CMS，常见的解决是：

减小 -XX:InitiatingHeapOccupancyPercent 提前启动标记周期。

**Mixed GC：当老年代空间达到阈值会触发 Mixed GC，选定所有新生代里的 Region，**

**根据全局并发标记阶段(下面介绍到)统计得出收集收益高的若干老年代 Region**

最后简单归纳一下：

G1把内存分成一块块的Region，每块的Region的大小都是一样的。

G1保留了YGC并加上了一种全新的MIXGC用于收集老年代。G1中没有Full GC，G1中的Full GC是采用serial old Full GC。在MIXGC中的Cset是选定所有young gen里的region，外加根据global concurrent marking统计得出收集收益高的若干old gen region。在YGC中的Cset是选定所有young gen里的region。通过控制young gen的region个数来控制young GC的开销。YGC与MIXGC都是采用多线程复制清除，整个过程会STW。

G1的低延迟原理在于其回收的区域变得精确并且范围变小了。

全局并发标记分的五个阶段。

用STAB来维持并发GC的准确性。

使用G1的最佳实践

G1 GC日志打印

<https://zhuanlan.zhihu.com/p/52841787>