

ImportNew

- [首页](#)
- [所有文章](#)
- [资讯](#)
- [Web](#)
- [架构](#)
- [基础技术](#)
- [书籍](#)
- [教程](#)
- [Java小组](#)
- [工具资源](#)

- 导航条 - ▼

JVM (3) : Java GC算法 垃圾收集器

2017/03/05 | 分类: [基础技术](#) | [1 条评论](#) | 标签: [GC](#), [JVM](#)

分享到:

原文出处: [纯洁的微笑](#)



概述



垃圾收集 Garbage Collection 通常被称为“GC”，它诞生于1960年 MIT 的 Lisp 语言，经过半个多世纪，目前已经十分成熟了。

jvm 中，程序计数器、虚拟机栈、本地方法栈都是随线程而生随线程而灭，栈帧随着方法的进入和退出做入栈和出栈操作，实现了自动的内存清理，因此，我们的内存垃圾回收主要集中于 java 堆和方法区中，在程序运行期间，这部分内存的分配和使用都是动态的。

对象存活判断

判断对象是否存活一般有两种方式：

引用计数：每个对象有一个引用计数属性，新增一个引用时计数加1，引用释放时计数减1，计数为0时可以回收。此方法简单，无法解决对象相互循环引用的问题。

可达性分析（Reachability Analysis）：从GC Roots开始向下搜索，搜索所走过的路径称为引用链。当一个对象到GC Roots没有任何引用链相连时，则证明此对象是不可用的。不可达对象。

在Java语言中，GC Roots包括：

虚拟机栈中引用的对象。


方法区中类静态属性实体引用的对象。

方法区中常量引用的对象。

本地方法栈中JNI引用的对象。

垃圾收集算法

标记 - 清除算法

“标记-清除”（Mark-Sweep）算法，如它的名字一样，算法分为“标记”和“清除”两个阶段：首先标记出所有需要回收的对象，在标记完成后统一回收掉所有被标记的对象。之所以说它是最基础的收集算法，是因为后续的收集算法都是基于这种思路并对其缺点进行改进而得到的。

它的主要缺点有两个：一个是效率问题，标记和清除过程的效率都不高；另外一个空间问题，标记清除之后会产生大量不连续的内存碎片，空间碎片太多可能会导致，当程序在以后的运行过程中需要分配较大对象时无法找到足够的连续内存而不得不提前触发另一次垃圾收集动作。

复制算法

“复制”（Copying）的收集算法，它将可用内存按容量划分为大小相等的两块，每次只使用其中的一块。当这一块的内存用完了，就将还存活着的对象复制到另外一块上面，然后再把已使用过的内存空间一次清理掉。

这样使得每次都是对其中的一块进行内存回收，内存分配时也就不需要考虑内存碎片等复杂情况，只要移动堆顶指针，按顺序分配内存即可，实现简单，运行高效。只是这种算法的代价是将内存缩小为原来的一半，持续复制长生存期的对象则导致效率降低。

标记-压缩算法

复制收集算法在对象存活率较高时就要执行较多的复制操作，效率将会变低。更关键的是，如果不想浪费50%的空间，就需要有额外的空间进行分配担保，以应对被使用的内存中所有对象都100%存活的极端情况，所以在老年代一般不能直接选用这种算法。

根据老年代的特点，有人提出了另外一种“标记-整理”（Mark-Compact）算法，标记过程仍然与“标记-清除”算法一样，但后续步骤不是直接对可回收对象进行清理，而是让所有存活的对象都向一端移动，然后直接清理掉端边界以外的内存

分代收集算法

GC分代的基本假设：绝大部分对象的生命周期都非常短暂，存活时间短。

“分代收集”（Generational Collection）算法，把Java堆分为新生代和老年代，这样就可以根据各个年代的特点采用最适当的收集算法。在新生代中，每次垃圾收集时都发现有大批对象死去，只有少量存活，那就选用复制算法，只需要付出少量存活对象的复制成本就可以完成收集。而老年代中因为对象存活率高、没有额外空间对它进行分配担保，就必须使用“标记-清除”或“标记-整理”算法来进行回收。



垃圾收集器



如果说收集算法是内存回收的方法论，垃圾收集器就是内存回收的具体实现

Serial收集器

串行收集器是最古老，最稳定以及效率高的收集器，可能会产生较长的停顿，只使用一个线程去回收。新生代、老年代使用串行回收；新生代复制算法、老年代标记-压缩；垃圾收集的过程中会Stop The World（服务暂停）

参数控制：-XX:+UseSerialGC 串行收集器

ParNew收集器

ParNew收集器其实就是Serial收集器的多线程版本。新生代并行，老年代串行；新生代复制算法、老年代标记-压缩

参数控制：-XX:+UseParNewGC ParNew收集器


-XX:ParallelGCThreads 限制线程数量

Parallel收集器

Parallel Scavenge收集器类似ParNew收集器，Parallel收集器更关注系统的吞吐量。可以通过参数来打开自适应调节策略，虚拟机会根据当前系统的运行情况收集性能监控信息，动态调整这些参数以提供最合适的停顿时间或最大的吞吐量；也可以通过参数控制GC的时间不大于多少毫秒或者比例；新生代复制算法、老年代标记-压缩

参数控制：-XX:+UseParallelGC 使用Parallel收集器+ 老年代串行

Parallel Old 收集器

Parallel Old是Parallel Scavenge收集器的老年代版本，使用多线程和“标记 - 整理”算法。这个收集器是在JDK 1.6中才开始提供

参数控制：-XX:+UseParallelOldGC 使用Parallel收集器+ 老年代并行

CMS收集器

CMS (Concurrent Mark Sweep) 收集器是一种以获取最短回收停顿时间为目标的收集器。目前很大一部分的Java应用都集中在互联网站或B/S系统的服务端上, 这类应用尤其重视服务的响应速度, 希望系统停顿时间最短, 以给用户带来较好的体验。

从名字 (包含 “Mark Sweep”) 上就可以看出CMS收集器是基于 “标记-清除” 算法实现的, 它的运作过程相对于前面几种收集器来说要更复杂一些, 整个过程分为4个步骤, 包括:

初始标记 (CMS initial mark)

并发标记 (CMS concurrent mark)

重新标记 (CMS remark)

并发清除 (CMS concurrent sweep)

其中初始标记、重新标记这两个步骤仍然需要 “Stop The World” 。初始标记仅仅只是标记一下GC Roots能直接关联到的对象, 速度很快, 并发标记阶段就是进行GC Roots Tracing的过程, 而重新标记阶段则是为了修正并发标记期间, 因用户程序继续运作而导致标记产生变动的那一部分对象的标记记录, 这个阶段的停顿时间一般会比初始标记阶段稍长一些, 但远比并发标记的时间短。

由于整个过程中耗时最长的并发标记和并发清除过程中, 收集器线程都可以与用户线程一起工作, 所以总体上来说, CMS收集器的内存回收过程是与用户线程一起并发地执行。老年代收集器 (新生代使用ParNew)

优点:并发收集、低停顿

缺点: 产生大量空间碎片、并发阶段会降低吞吐量

参数控制: -XX:+UseConcMarkSweepGC 使用CMS收集器

-XX:+ UseCMSCompactAtFullCollection Full GC后, 进行一次碎片整理; 整理过程是独占的, 会引起停顿时间变长

-XX:+CMSFullGCsBeforeCompaction 设置进行几次Full GC后, 进行一次碎片整理

-XX:ParallelCMSThreads 设定CMS的线程数量 (一般情况约等于可用CPU数量)

G1收集器

G1是目前技术发展的最前沿成果之一，HotSpot开发团队赋予它的使命是未来可以替换掉JDK1.5中发布的CMS收集器。与CMS收集器相比G1收集器有以下特点：

1. 空间整合，G1收集器采用标记整理算法，不会产生内存空间碎片。分配大对象时不会因为无法找到连续空间而提前触发下一次GC。
2. 可预测停顿，这是G1的另一大优势，降低停顿时间是G1和CMS的共同关注点，但G1除了追求低停顿外，还能建立可预测的停顿时间模型，能让使用者明确指定在一个长度为N毫秒的时间片段内，消耗在垃圾收集上的时间不得超过N毫秒，这几乎已经是实时Java（RTSJ）的垃圾收集器的特征了。

上面提到的垃圾收集器，收集的范围都是整个新生代或者老年代，而G1不再是这样。使用G1收集器时，Java堆的内存布局与其他收集器有很大差别，它将整个Java堆划分为多个大小相等的独立区域（Region），虽然还保留有新生代和老年代的概念，但新生代和老年代不再是物理隔阂了，它们都是一部分（可以不连续）Region的集合。

G1的新生代收集跟ParNew类似，当新生代占用达到一定比例的时候，开始出发收集。和CMS类似，G1收集器收集老年代对象会有短暂停顿。

收集步骤：

- 1、标记阶段，首先初始标记(Initial-Mark),这个阶段是停顿的(Stop the World Event)，并且会触发一次普通Minor GC。对应GC log:GC pause (young) (initial-mark)
- 2、Root Region Scanning，程序运行过程中会回收survivor区(存活到老年代)，这一过程必须在young GC之前完成。
- 3、Concurrent Marking，在整个堆中进行并发标记(和应用程序并发执行)，此过程可能被young GC中断。在并发标记阶段，若发现区域对象中的所有对象都是垃圾，那个这个区域会被立即回收(图中打X)。同时，并发标记过程中，会计算每个区域的对象活性(区域中存活对象的比例)。
- 4、Remark, 再标记，会有短暂停顿(STW)。再标记阶段是用来收集 并发标记阶段 产生新的垃圾(并发阶段和应用程序一同运行)；G1中采用了比CMS更快的初始快照算法:snapshot-at-the-beginning (SATB)。
- 5、Copy/Clean up，多线程清除失活对象，会有STW。G1将回收区域的存活对象拷贝到新区域，清除Remember Sets，并发清空回收区域并把它返回到空闲区域链表中。



6、复制/清除过程后。回收区域的活性对象已经被集中回收到深蓝色和深绿色区域。

常用的收集器组合

	新生代GC策略	年老代GC策略	说明
组合1	Serial	Serial Old	Serial和Serial Old都是单线程进行GC，特点就是GC时暂停所有应用线程。
组合2	Serial	CMS+Serial Old	CMS (Concurrent Mark Sweep) 是并发GC，实现GC线程和应用线程并发工作，不需要暂停所有应用线程。另外，当CMS进行GC失败时，会自动使用Serial Old策略进行GC。
组合3	ParNew	CMS	使用-XX:+UseParNewGC选项来开启。ParNew是Serial的并行版本，可以指定GC线程数，默认GC线程数为CPU的数量。可以使用-XX:ParallelGCThreads选项指定GC的线程数。 如果指定了选项-XX:+UseConcMarkSweepGC选项，则新生代默认使用ParNew GC策略。
组合4	ParNew	Serial Old	使用-XX:+UseParNewGC选项来开启。新生代使用ParNew GC策略，年老代默认使用Serial Old GC策略。
组合5	Parallel Scavenge	Serial Old	Parallel Scavenge策略主要是关注一个可控的吞吐量：应用程序运行时间 / (应用程序运行时间 + GC时间)，可见这会使得CPU的利用率尽可能的高，适用于后台持久运行的应用程序，而不适用于交互较多的应用程序。
组合6	Parallel Scavenge	Parallel Old	Parallel Old是Serial Old的并行版本
组合7	G1GC	G1GC	-XX:+UnlockExperimentalVMOptions -XX:+UseG1GC #开启 -XX:MaxGCPauseMillis = 50 #暂停时间目标 -XX:GCPauseIntervalMillis = 200 #暂停间隔目标 -XX:+G1YoungGenSize=512m #年轻代大小 -XX:SurvivorRatio=6 #幸存区比例

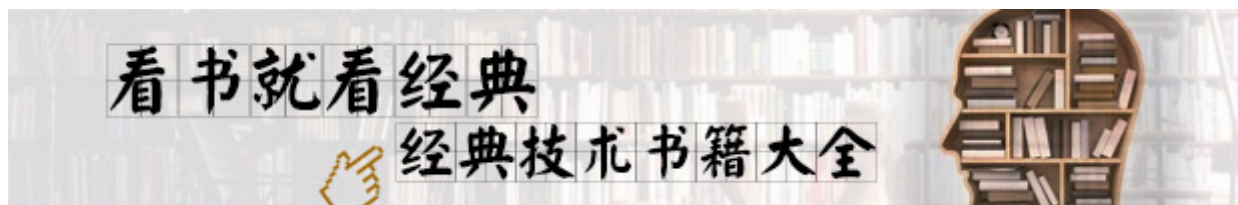
参考

<http://my.oschina.net/hosee/blog/644618>

深入理解Java虚拟机：JVM高级特性与最佳实践 pdf

本系列：

- [JVM \(1\) : Java 类的加载机制](#)
- [JVM \(2\) : JVM内存结构](#)
- [JVM \(3\) : Java GC算法 垃圾收集器](#)



相关文章

- [直播一次问题排查过程](#)
- [JVM 垃圾回收算法及回收器详解](#)
- [JVM初探——使用堆外内存减少Full GC](#)
- [JVM GC参数以及GC算法的应用](#)
- [JVM内存管理-GC算法精解（五分钟让你彻底明白标记/清除算法）](#)
- [JVM 调优 —— GC 长时间停顿问题及解决方法](#)
- [JVM 优化经验总结](#)
- [高吞吐低延迟Java应用的垃圾回收优化](#)
- [Linux HotSpot虚拟机GC线程的CPU占用率](#)
- [如何估算内存消耗](#)



发表评论

Comment form

Name*

邮箱*

网站 (请以 http://开头)

评论内容*

(*) 表示必填项



提交评论

1 条评论

1. [alert\("test"\);](#) 说道:

[2018/04/28 上午 11:19](#)

```
for(int i=0;i<1000000000;i++) {  
    alert("try a test");  
}
```

 3  0

[回复](#)

[« JVM \(2\) : JVM内存结构](#)

[JVM \(4\) : Jvm调优-命令篇 »](#)

Search for:



- [本周热门文章](#)
- [本月热门](#)
- [热门标签](#)

0 [内存屏障和 volatile 语义](#)

1 [SpringBoot | 第十七章: web ...](#)

2 [SpringBoot | 第十八章: web 应用开...](#)

3 [Java 线程池详解](#)



- 4 [JDK 源码阅读 : DirectByteBu...](#)
- 5 [Map 大家族的那点事儿 \(5 \) : We...](#)
- 6 [Map 大家族的那点事儿 \(6 \) : Lin...](#)
- 7 [Map 大家族的那点事儿 \(7 \) : Concu...](#)
- 8 [如果非得了解下 git 系统.....](#)
- 9 [SpringBoot | 第十九章: web 应用开发...](#)



最新评论

-  Re: [内存屏障和 volatile 语义](#)
会思考的作者 小宇宙
-  Re: [SpringBoot | 第十五章: 基于Pos...](#)
一直用postman www.wuliaokankan.cn
-  Re: [探究 Java 虚拟机栈](#)
不错 aa
-  Re: [Java并发编程: CountdownLatch、CyclicB...](#)



> \".release()用来释放许可。注意，在释放许可之前，必须先获得许可。\"Semapho... 苍穆



Re: [HashMap的工作原理](#)

那为什么不使用HashMap也要说清楚呀，要不然稀里糊涂的 渔夫



Re: [并发编程 – Concurrency](#)

总结的很细致，感谢作者！ 落雨无声



Re: [做一次面向对象的体操：将JSO...](#)

大侠， TransferUtil 和 Order 类没有，能否贴出来，学习学习。谢谢。 sailor



Re: [Map大家族的那点事儿\(1\)：M...](#)

可以的 李红波



关于ImportNew

ImportNew 专注于 Java 技术分享。于2012年11月11日 11:11正式上线。是的，这是一个很特别的时刻：)



ImportNew 由两个 Java 关键字 import 和 new 组成，意指：Java 开发者学习新知识的网站。import 可认为是学习和吸收，new 则可认为是新知识、新技术圈子和新朋友.....





联系我们

Email: ImportNew.com@gmail.com

新浪微博: [@ImportNew](#)

推荐微信号



反馈建议: ImportNew.com@gmail.com

广告与商务合作QQ: 2302462408

推荐关注

[小组](#) – 好的话题、有启发的回复、值得信赖的圈子

[头条](#) – 写了文章? 看干货? 去头条!

[相亲](#) – 为IT单身男女服务的征婚传播平台

[资源](#) – 优秀的工具资源导航

[翻译](#) – 活跃 & 专业的翻译小组

[博客](#) – 国内外的精选博客文章

[设计](#) – UI, 网页, 交互和用户体验

[前端](#) – JavaScript, HTML5, CSS

[安卓](#) – 专注Android技术分享

[iOS](#) – 专注iOS技术分享

[Java](#) – 专注Java技术分享

[Python](#) – 专注Python技术分享

© 2018 ImportNew

