链滴

初步认识 Garbage First（G1）垃圾回收器

作者：superstonne

原文链接：https://ld246.com/article/1565761967329

来源网站：链滴

本文将会介绍 Garbage First（简称 G1） 垃圾回收器的基本使用，以及在 HotSpot 中如何应用 G1 通过本文，你将会学习到 G1 的内部工作原理、它的关键参数，以及如何读懂它产生的日志信息。

鉴于本文目标读者是有一定的 Java 开发经验的程序员，因此本文不再累赘 Java 语言、Java 开发环 。略过这些部分，我们直接进入主题 JVM——Java虚拟机。

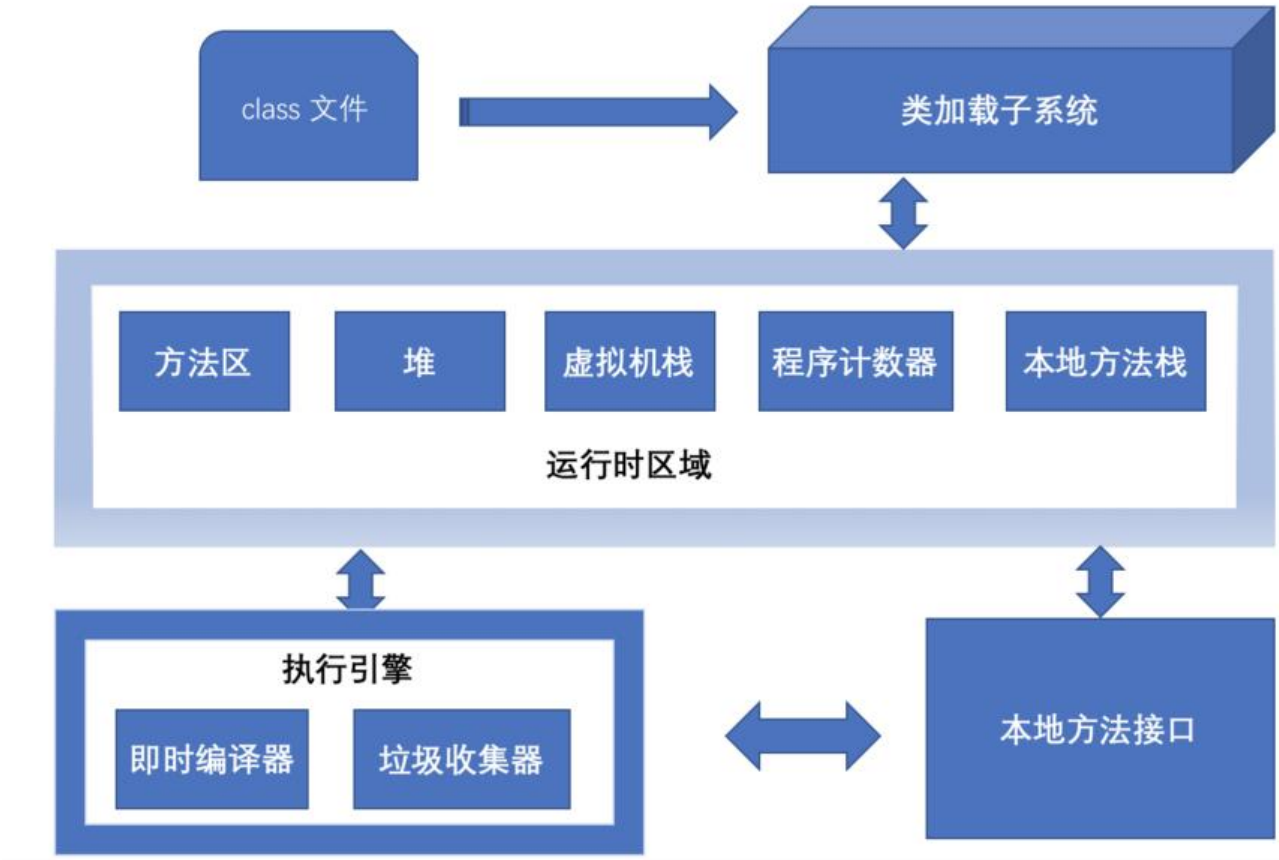
Java Virtual Machine（Java 虚拟机）

JVM 其实就是运行在操作系统上层的一个软件，但是相对于 Java 程序来说，又可以理解它为一个 CP ，负责运行 Java 程序。Java 程序实现了一系列的接口和方法运行于 JVM 之上，每一款特定的 JVM 例如运行在 Windows、OS X、Android、Linux 等不同硬件设备的操作系统上） 负责与底层的操作 统交互，因此 Java 语言依赖于 JVM 实现了跨平台。

虽然 JVM 运行 Java 程序，但是其实它本身并不认识 Java 语言，它只认得二进制内容的 class 文件。 lass 文件中包含了 JVM 指令、符号表（用于保存有关源程序构造的各种信息的数据结构），以及其 的辅助信息。得益于 JVM 是靠运行字节码指令来工作的，市面上就出现了很多运行在 JVM 之上的编 语言，例如：Kotlin、Scala、Clojure、Groovy、Jython、JRuby、Ceylon、Eta、Haxe 等。

Hotspot JVM 架构

Hotspot JVM 的体系结构性能高，并且扩展能力强。主要表现在它的即时编译器 JIT Compiler：它 依据程序的运行，动态地将部分代码编译为本地机器指令来提升程序的运行效率。同时得益于它的多 程垃圾回收器，使得它可以在大机器上也运行得游刃有余。



从上图中我们可以看到，Hotspot JVM 主要包括：

- 类加载系统
- 运行时区域
- 执行引擎
- 本地方法接口

性能优化相关的模块，主要有运行时区域的堆区、执行引擎中的即时编译器和垃圾回收器。

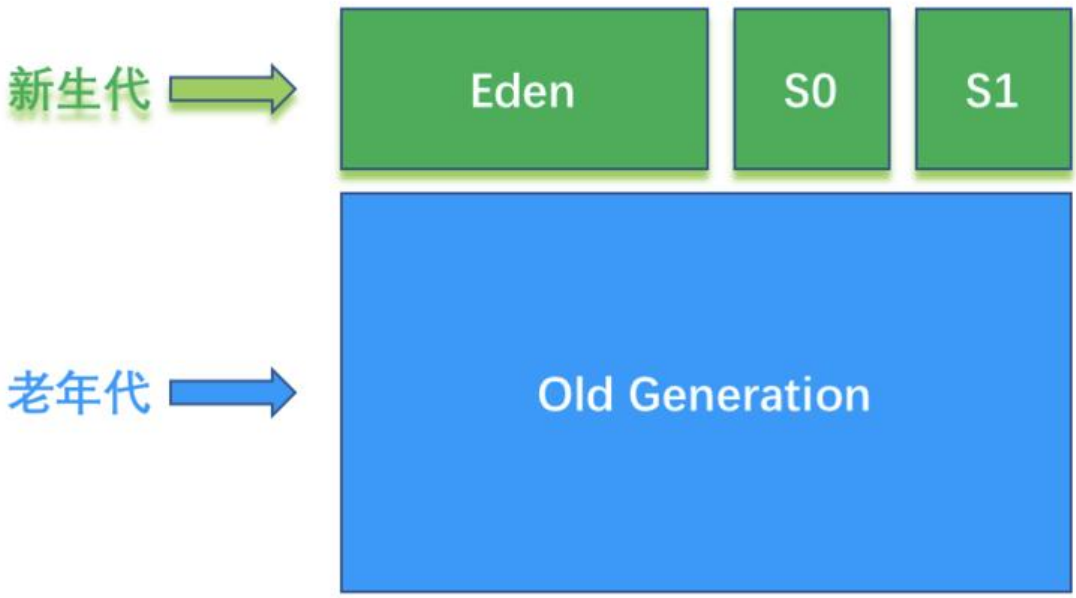
这三块中，即时编译器虽然对性能也有很大的影响，但是在最近的 JVM 版本中，这一块可优化的程非常有限。因此对于Java 程序员来说，JVM 的性能优化主要就在堆区和垃圾回收器上。

回顾 CMS（Concurrent Mark Sweep）垃圾回收器

CMS 与应用线程并行工作来达到低停顿目标。但是由于它在老年代采用了标记清除的垃圾回收算法在垃圾回收过程中，它不会迁移依旧存活的对象，因此会出现内存碎片问题。

CMS 垃圾回收阶段详解

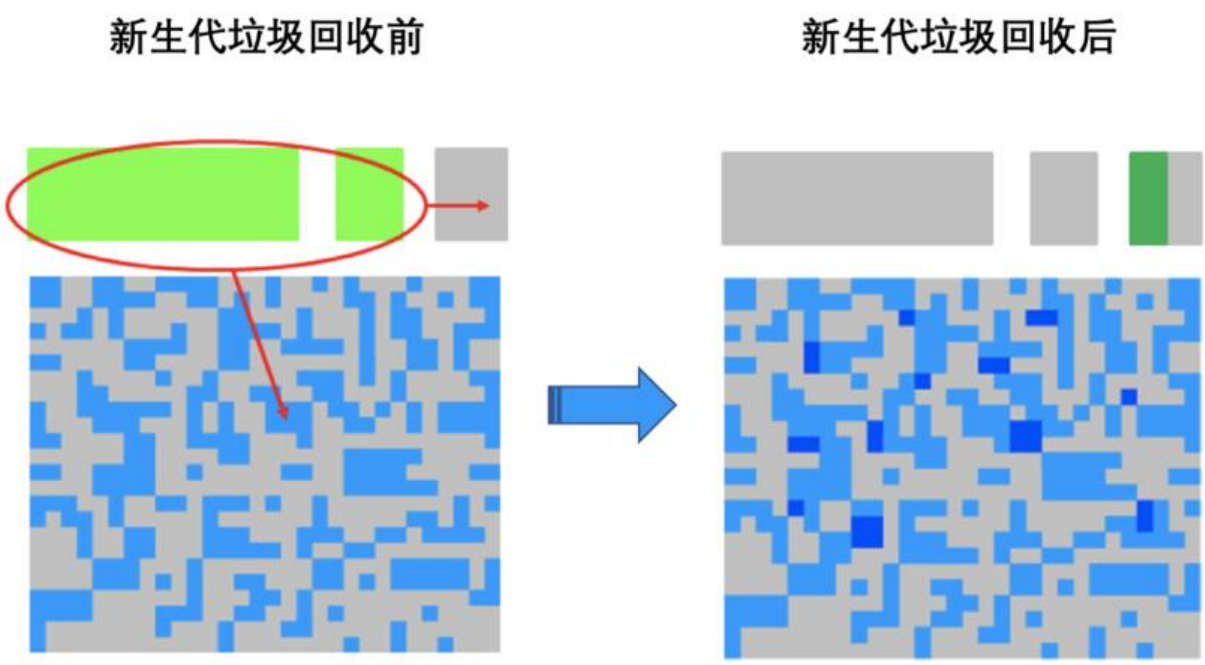
CMS 的堆内存结构分了二块：老年代、新生代。新生代又被分成了二块：Eden 和 Survivors 区域。创建的对象一般会分配在 Eden 区域。



CMS 新生代垃圾回收

CMS 在新生代采取复制清除的垃圾回收算法。当 Eden 区域内存空间不足时候，会发生新生代的垃圾回收。如图中所示，新生代垃圾回收会将存活的对象分别移动到 To Survivor 区域或者部分 From Survisor 区域的对象移动到老年代（对象的年龄到达一定的年龄，由 JVM 参数控制）。

新生代垃圾回收过后，如下图深绿色部分为 Eden 区域存活的对象移动到了 To Survisor 区域。老年中深蓝色的部分为 From Survisor 晋升到老年代的对象（年纪大的对象：经历过一次垃圾回收，年龄大一岁）。

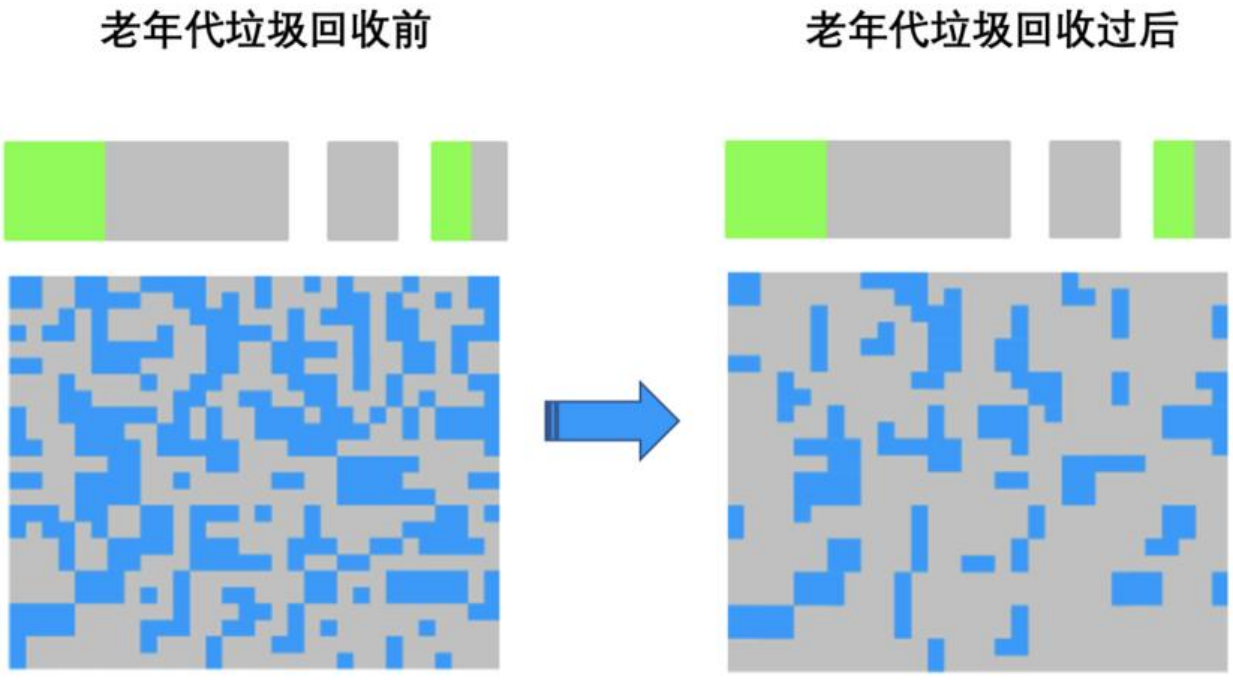


深绿色：Eden 区域存活对象迁移到了 Survivor 区域
深蓝色：新生代对象晋升到了老年代

CMS 老年代垃圾回收

CMS 老年代采取标记清除的垃圾回收算法（除非老年代产生碎片问题时候，无法在容纳大对象的时，会发生压缩）。具体在下面老年垃圾回收的步骤中分析清楚，在此不累赘。

如下图，即为一次老年代垃圾回收过的场景，可见存活的对象没有被移动，只是死亡的对象已经被清。



原文链接：初步认识 Garbage First（G1）垃圾回收器

CMS 老年代垃圾回收的几个阶段

初始标记（Stop the World）

该阶段 CMS 会停止应用线程，标记出 GC ROOT 直接关联的对象，因为直接关联的对象较少，所以阶段虽然造成了 GC 停顿，但是时间较短。

并行标记

该阶段 CMS 与应用线程并行运行，CMS 遍历第一阶段标记出来的存活对象，找出老年代中所有可以关联到的存活对象。

注意：在该阶段和并行清除阶段，应用程序始终在运行，则意味着不断会有新的对象产生，这些新的象在产生时候会被立即标记为存活对象。

重新标记（Stop the World）

该阶段 CMS 会停止应用线程，因为上个阶段的并行标记，某些对象可能会在 CMS 跟踪过后被应用程更新为可达状态，或者更新到非可达状态，因此要进行重新标记。

并行清除

该阶段会清理掉在标记阶段被确认为不可达对象的内存空间。

G1 垃圾回收器

G1 简述

在 Oracle JDK7 的第四个版本开始，以及之后的发行版本中都支持 G1 垃圾回收器。G1 专门为大内的多核机器而设计，它的诞生使得我们的应用程序可以在满足高吞吐量的同时也可以达到低停顿。它要有如下优点：

它可以与应用程序线程并行运行

不需要长的 GC 停顿就可以完成内存空间压缩，无碎片化问题（CMS 的内存碎片问题）

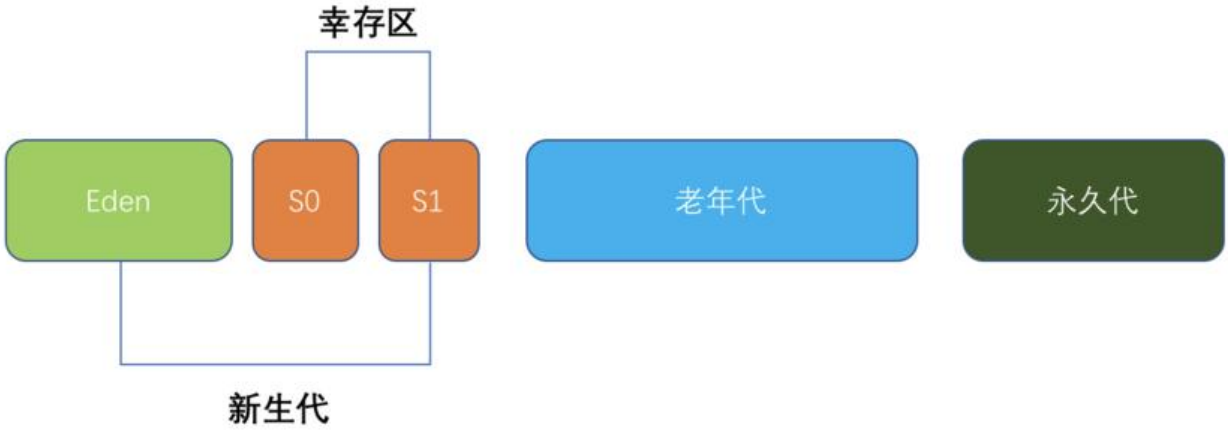
GC 的暂停时间可以由用户来控制

吞吐量高

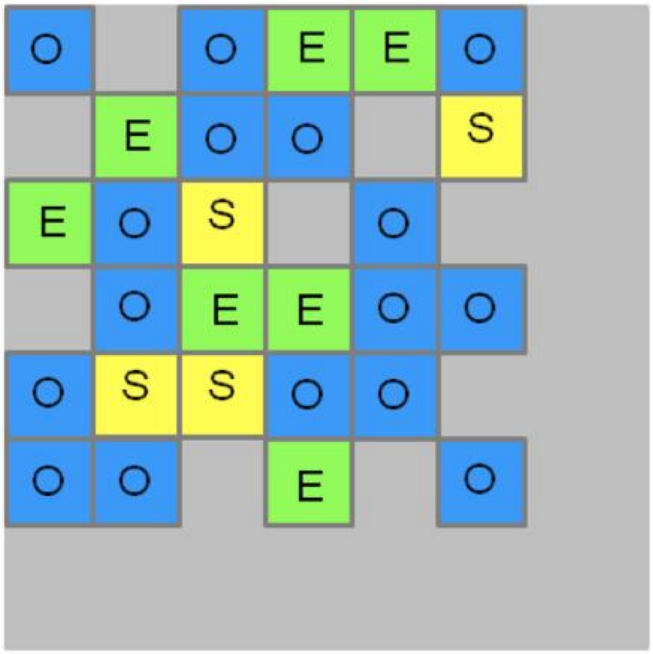
相比较于 CMS 垃圾回收器，G1 是一款压缩型的垃圾回收器，同时 G1 的内存结构由大量的块区域组。这样使得垃圾回收变得简单，同时降低了 CMS 的碎片化问题。得益于优秀的内存结构设计，在垃回收变得简单的同时回收性能也有了很大的提升，因此 G1 也被计划为 CMS 的长期替代品。

G1 内存结构

在 G1 之前的垃圾回收器里，都将内存分成了三个部分：新生代（新生代又包括 Eden 和 Survivor）老年代、永久代。如下图所示：



G1 采取了一种不同的内存结构。G1 将内存划分成了若干个相等大小的内存块，每个内存块会是三种型中一种（Eden、Survivor、Old）。



E：新生代 Eden 区域
S：新生代 Survivor 区域
O：老年代区域

G1 采取和 CMS 同样的方式进行对象标记，通过一次全局标记后，它就知道哪些区块中存在了大量垃圾对象。接下来它就在这些垃圾对象较多的区块上做垃圾回收，这样可以达到一个高的回收率。

这也是为什么 G1 名称的由来（Garbage First）。在垃圾回收时候，G1 会将这些区域的存活对象移

到一个空白的区块中，同时旧的区块将被释放，这样就达到了压缩的效果，消除了内存碎片问题，这操作是和应用程序的线程并行工作的，这样降低了 GC 停顿时间，提升了应用程序的吞吐量。

我们需要注意的是，用户可以指定 G1 停顿时间的目标，但是 G1 不会立刻达到这个目标。G1 会通一次次的垃圾回收，依赖以往的回收数据来不断地改变每次回收的区块数量来改变停顿时间，最终达我们设置的目标。

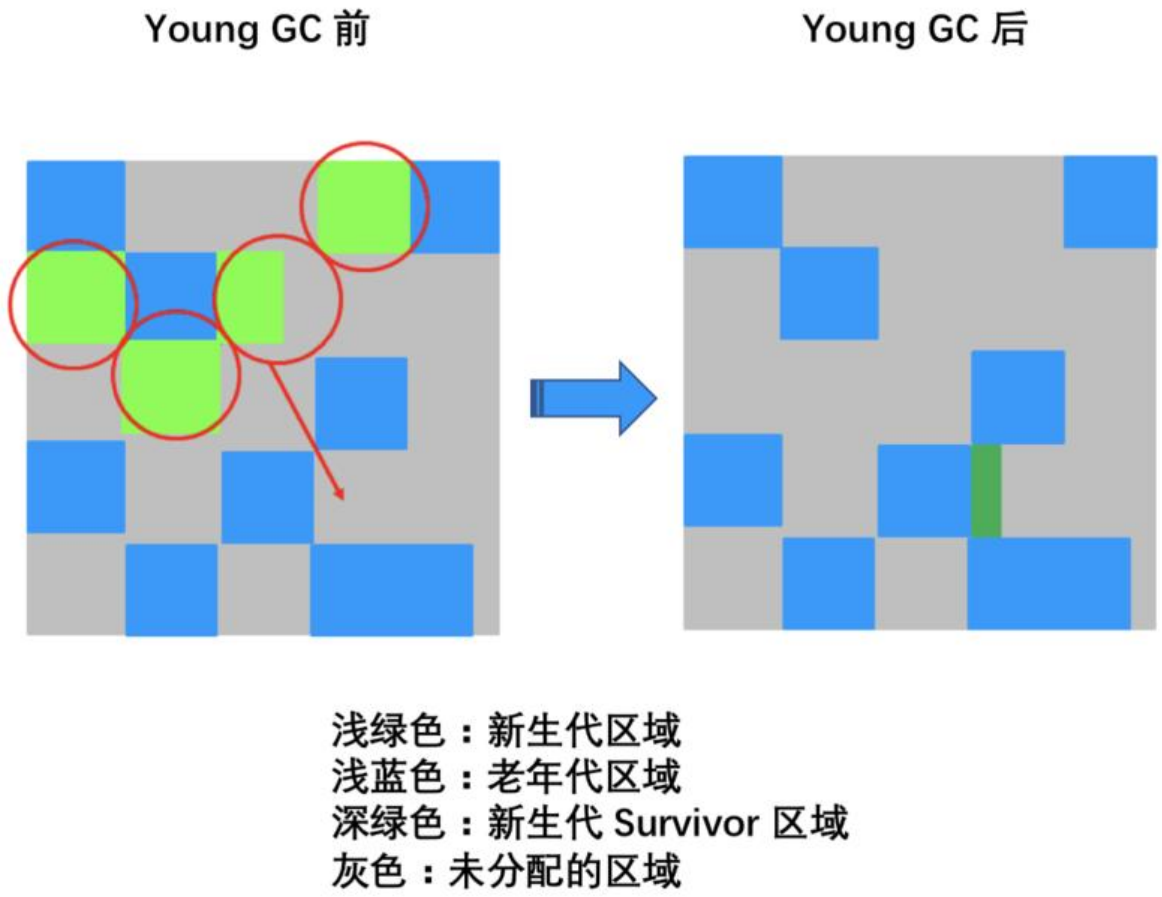
图解 G1 垃圾回收

G1 的堆内存在 JVM 启动的时候将被分割成 2000 块左右相同大小（1~32M）的区域，这些区域会应到传统的 JVM 的 Eden、Survivor、Old 三种区域中的一种。假如你有一个很大的对象，它所需要用的内存在单个区域的50%以上，这样的对象会被安置在大对象区域（有若干个连续的区域组成一个对象区域）。因为垃圾回收的单位以区域来进行，因此垃圾回收可以并行的与应用程序同时进行。

G1 的 Young GC 如何进行

新生代的存活对象会被迁移到一个或者多个 Survivor 区域，当然如果某些对象的年龄已经到达了晋的年龄，他们会被迁移到老年代区域。这里我们需要注意的是，这里会有一个 GC 停顿（Stop the W rld），因为 G1 需要知道此时 Eden 和 Survivor 区域的大小，这些信息会被记录来以优化之后的 G 、G1 会不断的调整单个区域的大小，来达到我们设置 GC 停顿指标。

当经过一次 Young GC 之后，内存将会变成如下的模样，新生代存活的对象被压缩到了图中深绿色区域中。旧的垃圾区域内存已经被 G1 释放掉了，以备新的对象分配使用。



由此我们可以得出如下关键点：

Young GC 是会有 STW 事件，此时用户线程暂停

Young GC 时候 GC 线程是多线程并行回收的

老年代如何回收

初始标记

初始标记操作其实是混合在 Young GC 中进行的，这个阶段会标记出一些新生代的区域，这些被标出来的区域里面包含了指向老年代对象的引用。

并行标记

并行标记阶段 GC 线程和应用线程并行工作，该阶段会标记出没有对象存活的区域。

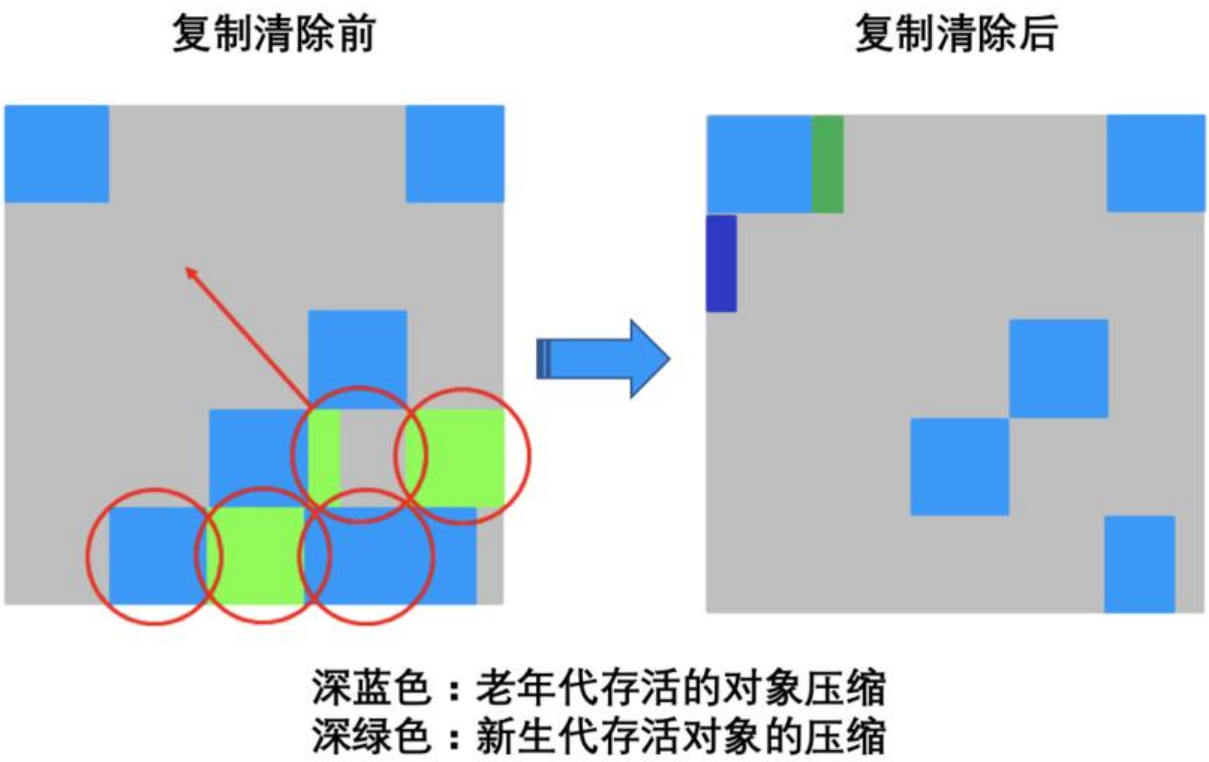
重新标记

重新标记阶段会立刻释放掉在上一阶段标记无存活对象的区域，并且这一阶段会计算出所有区域的对存活率。

复制清除

复制清除阶段是和 Young GC 的一个混合阶段，这个阶段会选出对象存活率较低的区域来进行垃圾收，因为这样垃圾回收的效率也会更好。

在复制清除阶段过后，堆内存会变成如下图所示，新生代的对象会被压缩至深绿色区域，老年代的对会被压缩至深蓝色区域。



由此我们可以得出老年代垃圾回收的几个关键点：

并行标记阶段

计算每个区域的对象存活率（与应用线程并行）

找出最合适的用来回收垃圾的区域

重新标记阶段

回收上一阶段标记出的没有存活对象的区域

复制清除阶段

新生代，老年代（重新标记阶段选举出的对象存活率较低的区域）混合进行

G1 的主要参数

-XX:+UseG1GC

告诉 JVM 使用 G1 垃圾回收器来进行垃圾回收。

-XX:MaxGCPauseMillis=200

设置 GC 的停顿目标为 200 毫秒，正如我们上面所说，这个目标是在不断地优化下达成的，并不是立就会达到这个目标。

-XX:InitiatingHeapOccupancyPercent=45

告诉 G1 当堆里面的内存占用率等于或者高于 45% 时候开始一个 GC 周期。

-XX:G1HeapRegionSize=n

单个区域的大小，通常在 1~32M 之间，如果不被指定，则 JVM 启动时候会自行根据设置号的堆的大小进行计算，最终分出大约 2000 个左右的区域。

理解 G1 的日志

只有我们看懂了 GC 的日志，才能对症下药进行性能优化。所以接下来我们进行 G1 的日志解读。

设置 G1 打印 GC 日志

G1 共有三个级别的日志：fine、finer、finest。

-XX:+ PrintGC

设置 GC 的日志级别为 fine，该级别打印日志格式参考如下：

[GC pause (G1 Humongous Allocation) (young) (initial-mark) 24M- >21M(64M), 0.2349730 se s]

[GC pause (G1 Evacuation Pause) (mixed) 66M->21M(236M), 0.1625268 secs]

-XX:+PrintGCDetails

设置 GC 日志级别为 finer，该级别的日志会比上一级别更详细，例如：

每个阶段的最小，最大，平均耗时

显示 Eden，Survivor 和总堆大小

显示一些细小操作的耗时，例如选择/释放 CSet

显示 RSet 操作的日志

[Ext Root Scanning (ms): Avg: 1.7 Min: 0.0 Max: 3.7 Diff: 3.7]

[Eden: 818M(818M)->0B(714M) Survivors: 0B->104M Heap: 836M(4096M)->409M(4096M)]
-XX:+UnlockExperimentalVMOptions -XX:G1LogLevel=finest

设置日志级别为 finest，比起 finer 级别，该级别打印出了 worker 线程的日志：

[Ext Root Scanning (ms): 2.1 2.4 2.0 0.0
Avg: 1.6 Min: 0.0 Max: 2.4 Diff: 2.3]
[Update RS (ms): 0.4 0.2 0.4 0.0
Avg: 0.2 Min: 0.0 Max: 0.4 Diff: 0.4]
[Processed Buffers : 5 1 10 0
Sum: 16, Avg: 4, Min: 0, Max: 10, Diff: 10]
设置日志时间戳的格式
-XX:+PrintGCTimeStamps

该日志格式显示的当前时间距离 JVM 启动时候时间，例如：

1.729: [GC pause (young) 46M->35M(1332M), 0.0310029 secs]
-XX:+PrintGCDateStamps

该格式会显示当前的日志，例如：

2012-05-02T11:16:32.057+0200: [GC pause (young) 46M->35M(1332M), 0.0317225 secs]
日志解读
Parallel Time

414.557: [GC pause (young), 0.03039600 secs] [Parallel Time: 22.9 ms] [GC Worker Start (ms): 096.0 7096.0 7096.1 7096.1 706.1 7096.1 7096.1 7096.1 7096.2 7096.2 7096.2 7096.2 Avg: 709 .1, Min: 7096.0, Max: 7096.2, Diff: 0.2]

Parallel Time：GC 停顿时间，并行线程的耗时。

Worker Start：GC 线程开始工作时间（时间按照线程 ID 排序）。

External Root Scanning

[Ext Root Scanning (ms): 3.1 3.4 3.4 3.0 4.2 2.0 3.6 3.2 3.4 7.7 3.7 4.4
Avg: 3.8, Min: 2.0, Max: 7.7, Diff: 5.7]
扫描外部的 Root 对象耗时。

Update Remembered Set

[Update RS (ms): 0.1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 Avg: 0.0, Min: 0.0, Max: 0.1, Diff: 0 1]
[Processed Buffers : 26 0 0 0 0 0 0 0 0 0 0 0
Sum: 26, Avg: 2, Min: 0, Max: 26, Diff: 26]

更新 RSet 耗费的时间。

Scanning Remembered Sets

[Scan RS (ms): 0.4 0.2 0.1 0.3 0.0 0.0 0.1 0.2 0.0 0.1 0.0 0.0 Avg: 0.1, Min: 0.0, Max: 0.4, Diff: 0.3]

扫描 RSet 耗费的时间。

Object Copy

[Object Copy (ms): 16.7 16.7 16.7 16.9 16.0 18.1 16.5 16.8 16.7 12.3 16.4 15.7 Avg: 16.3, Min: 1
.3, Max: 18.1, Diff: 5.8]

迁移对象耗时，例如：Eden 存活的对象拷贝到 Survivor 区域，年龄达到晋升的对象，从 Survivor
贝到老年代。

Termination Time

[Termination (ms): 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 Avg: 0.0, Min: 0.0, Max: 0.0, Diff: 0.0] [Termination Attempts : 1 1 1 1 1 1 1 1 1 1 1 Sum: 1
, Avg: 1, Min: 1, Max: 1, Diff: 0]

GC 线程的停止时间，当 GC 线程处理完手头的任务时候会回归到停止池中等待新的工作，这段时间
是停止时间。

GC Worker End

[GC Worker End (ms): 7116.4 7116.3 7116.4 7116.3 7116.4 7116.3 7116.4 7116.4 7116.4 7116.4
7116.3 7116.3
    Avg: 7116.4, Min: 7116.3, Max: 7116.4, Diff: 0.1]
[GC Worker (ms): 20.4 20.3 20.3 20.2 20.3 20.2 20.2 20.2 20.3 20.2 20.1 20.1
     Avg: 20.2, Min: 20.1, Max: 20.4, Diff: 0.3]

GC worker end time：GC 线程停止的时间

GC worker time：GC 线程的工作时间

GC Worker Other

[GC Worker Other (ms): 2.6 2.6 2.7 2.7 2.7 2.7 2.7 2.8 2.8 2.8 2.8 2.8
    Avg: 2.7, Min: 2.6, Max: 2.8, Diff: 0.2]

GC 线程的非 GC 工作的耗时，例如 GC 线程的启动、暂停等时间，这些时间不算在 GC 垃圾回收的
间内。

Clear CT

[Clear CT: 0.6 ms]

清理 CardTable 耗时。（RSet 依赖 CardTable 记录当前区域的存活对象）

Other

[Other: 6.8 ms]

垃圾回收阶段衔接的耗时。

CSet

[Choose CSet: 0.1 ms]

选择那些即将要被进行垃圾回收的区域耗时。

Ref Proc

[Ref Proc: 4.4 ms]

处理软，弱等引用耗费的时间。

Ref Enq

[Ref Enq: 0.1 ms]

将软，弱引用关联到等待列表耗费的时间。

Free CSet

[Free CSet: 2.0 ms]

清理 CSet 包含的区域的垃圾耗费的时间。

总结

通过这篇文章，我们一起探讨了 Hotspot JVM 基础架构，同时回顾了 CMS 垃圾回收的工作过程；下来，我们学习了 G1 垃圾回收器的内存结构，以及 G1 垃圾回收过程；最后我们学习到了 G1 的关配置参数，以及 G1 垃圾回收日志。

至此，相信大家已经对 G1 有了初步的认识。俗话说得好，纸上得来终觉浅，需要深刻地理解 G1，们还是要多多实战，如果平时工作中用不到，我们可以自己编写一点小程序，然后对着日志，看看我是否能解读每一句日志的含义，这样纸上得来的知识就会慢慢地刻在脑海里，一起加油学习吧。

以上只是粗浅地认识了一下 G1 垃圾回收器，当然 G1 垃圾回收过程中还有很多细节没有提到，待我初步了解之后再深入地去学习。