回首页

G1 垃圾收集器介绍

创建时间: 2018-05-08 00:00:00

[TOC]

之前根据 Sun 的内存管理白皮书介绍了在 HotSpot JVM 分代算法中的几个垃圾收集器，本文将介绍 G1 垃圾收集器。

G1 的主要关注点在于达到**可控的停顿时间**，在这个基础上尽可能提高吞吐量，这一点非常重要。

G1 被设计用来长期取代 CMS 收集器，和 CMS 相同的地方在于，它们都属于并发收集器，在大部分的收集阶段都不需要挂起应用程序。区别在于，G1 没有 CMS 的碎片化问题（或者说不那么严重），同时提供了更加可控的停顿时间。

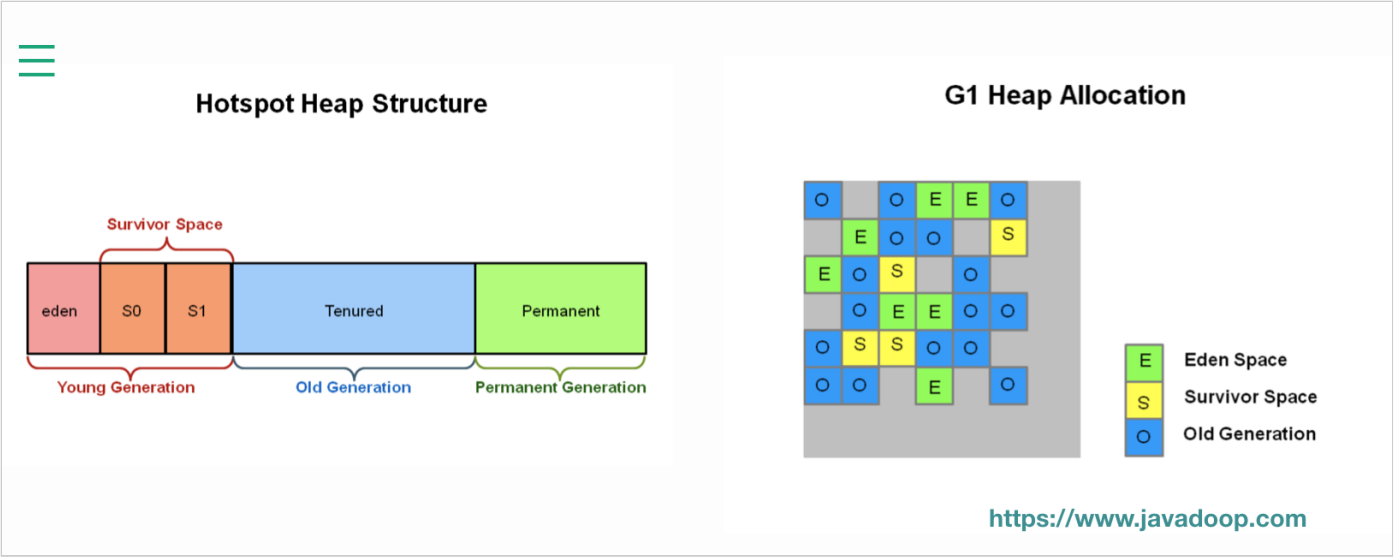
如果你的应用使用了较大的堆（如 6GB 及以上）而且还要求有较低的垃圾收集停顿时间（如 0.5 秒），那么 G1 是你绝佳的选择，是时候放弃 CMS 了。

阅读建议：本文力求用简单的话介绍清楚 G1 收集器，但是并不会重复介绍每一个细节，所以希望读者了解其他几个收集器的工作过程，尤其是 CMS 收集器。

G1 总览

首先是内存划分上，之前介绍的分代收集器将整个堆分为年轻代、老年代和永久代，每个代的空间是确定的。

而 G1 将整个堆划分为一个个大小相等的小块（每一块称为一个 region），每一块的内存是连续的。和分代算法一样，G1 中每个块也会充当 Eden、Survivor、Old 三种角色，但是它们不是固定的，这使得内存使用更加地灵活。



执行垃圾收集时，和 CMS 一样，G1 收集线程在标记阶段和应用程序线程**并发**执行，标记结束后，G1 也就知道哪些区块基本上是垃圾，存活对象极少，G1 会先从这些区块下手，因为从这些区块能很快释放得到很大的可用空间，**这也是为什么 G1 被取名为 Garbage-First 的原因**。

> 这里只不过是先介绍些概念，没看懂没关系，往下看

在 G1 中，目标停顿时间非常非常重要，用 -XX:MaxGCPauseMillis=200 指定期望的停顿时间。

G1 使用了**停顿预测模型**来满足用户指定的停顿时间目标，并基于目标来选择进行垃圾回收的区块数量。G1 采用增量回收的方式，每次回收一些区块，而不是整堆回收。

我们要知道 G1 不是一个实时收集器，它会尽力满足我们的停顿时间要求，但也不是绝对的，它基于之前垃圾收集的数据统计，估计出在用户指定的停顿时间内能收集多少个区块。

注意：G1 有和应用程序一起运行的并发阶段，也有 stop-the-world 的并行阶段。但是，**Full GC 的时候还是单线程运行的**，所以我们应该尽量避免发生 Full GC，后面我们也会介绍什么时候会触发 Full GC。

G1 内存占用

G1 比 ParallelOld 和 CMS 会需要更多的内存消耗，那是因为有部分内存消耗于簿记（accounting）上，如以下两个数据结构：

- **Remembered Sets**：每个区块都有一个 RSet，用于记录进入该区块的对象引用（如区块 A 中的对象引用了区块 B，区块 B 的 Rset 需要记录这个信息），它用于实现收集过程的并行化以及使得区块能进行独立收集。总体上 Remembered Sets 消耗的内存小于 5%。
- **Collection Sets**：将要被回收的区块集合。GC 时，在这些区块中的对象会被复制到其他区块中，总体上 Collection Sets 消耗的内存小于 1%。

G1 工作流程

前面啰里啰嗦说了挺多的，唯一要记住的就是，G1 的设计目标就是尽力满足我们的目标停顿时间上的要求。

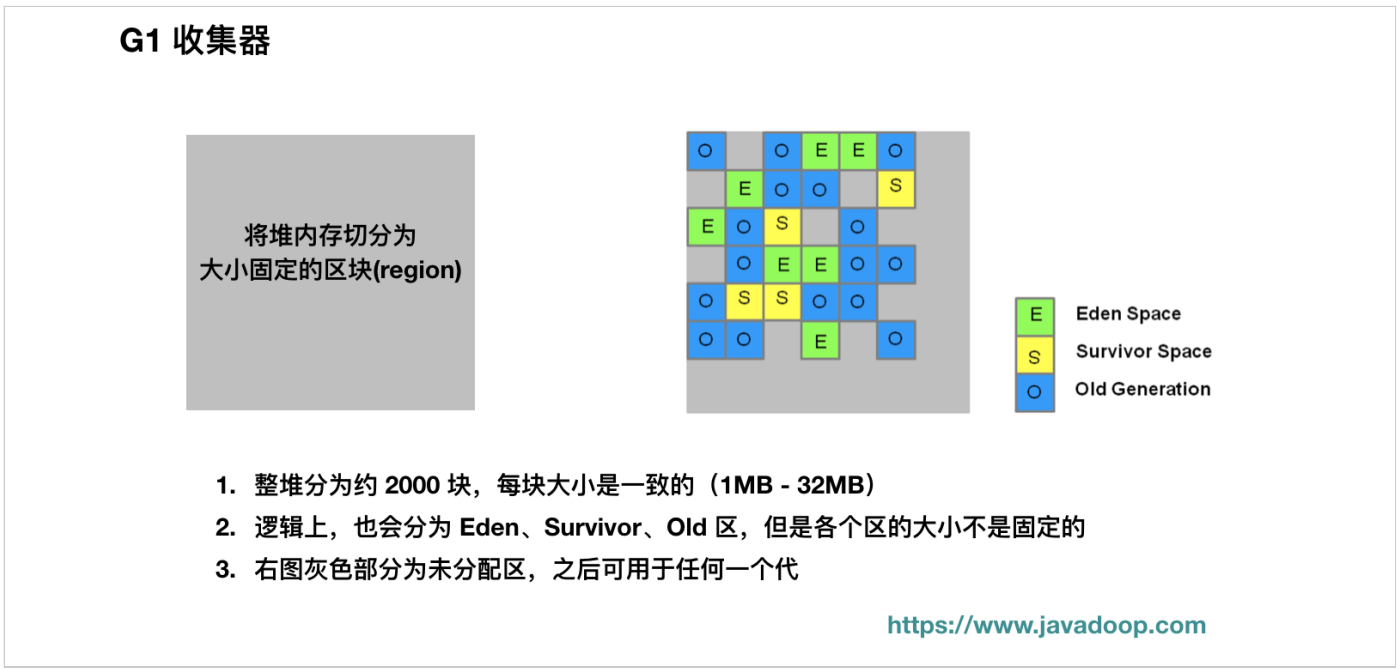
本节介绍 G1 的收集过程，G1 收集器主要包括了以下 4 种操作：

- 1、年轻代收集
- 2、并发收集，和应用线程同时执行
- 3、混合式垃圾收集
- *、必要时的 Full GC

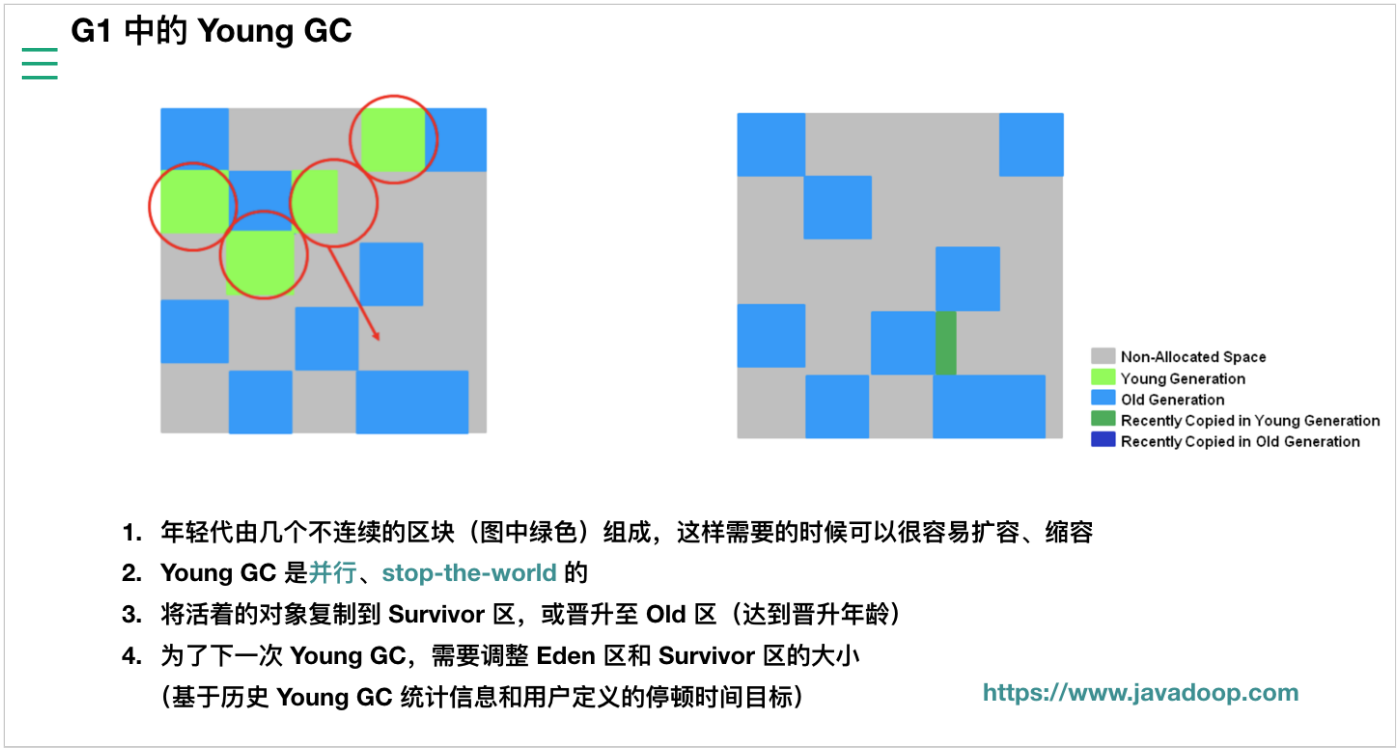
接下来，我们进行一一介绍。

年轻代收集

首先，我们来看下 G1 的堆结构：



年轻代中的垃圾收集流程（Young GC）：



我们可以看到，年轻代收集概念上和之前介绍的其他分代收集器大差不差的，但是它的年轻代会动态调整。

Old GC / 并发标记周期

接下来是 Old GC 的流程（含 Young GC 阶段），其实把 Old GC 理解为**并发周期**是比较合理的，不要单纯地认为是清理老年代的区块，因为这一步和年轻代收集也是相关的。下面我们介绍主要流程：

- 初始标记：stop-the-world，它伴随着一次普通的 Young GC 发生，然后对 Survivor 区（root region）进行标记，因为该区可能存在对老年代的引用。

  > 因为 Young GC 是需要 stop-the-world 的，所以并发周期直接重用这个阶段，虽然会增加 CPU 开销，但是停顿时间只是增加了一小部分。

- 扫描根引用区：因为先进行了一次 YGC，所以当前年轻代只有 Survivor 区有存活对象，它被称为根引用区。扫描 Survivor 到老年代的引用，该阶段必须在下一次 Young GC 发生前结束。

  > 这个阶段不能发生年轻代收集，如果中途 Eden 区真的满了，也要等待这个阶段结束才能进行 Young GC。

- 并发标记：寻找整个堆的存活对象，该阶段可以被 Young GC 中断。

  > 这个阶段是并发执行的，中间可以发生多次 Young GC，Young GC 会中断标记过程

- 重新标记：stop-the-world，完成最后的存活对象标记。使用了比 CMS 收集器更加高效的 snapshot-at-the-beginning (SATB) 算法。

  > Oracle 的资料显示，这个阶段会回收完全空闲的区块

- 清理：清理阶段真正回收的内存很少。

到这里，G1 的一个并发周期就算结束了，其实就是主要完成了垃圾定位的工作，定位出了哪些分区是垃圾最多的。因为整堆一般比较大，所以这个周期应该会比较长，中间可能会被多次 stop-the-world 的 Young GC 打断。

混合垃圾回收周期

并发周期结束后是混合垃圾回收周期，不仅进行年轻代垃圾收集，而且回收之前标记出来的老年代的垃圾最多的部分区块。

混合垃圾回收周期会持续进行，直到几乎所有的被标记出来的分区（垃圾占比大的分区）都得到回收，然后恢复到常规的年轻代垃圾收集，最终再次启动并发周期。

Full GC

到这里我们已经说了年轻代收集、并发周期、混合回收周期了，大家要熟悉这几个阶段的工作。

下面我们来介绍特殊情况，那就是会导致 **Full GC** 的情况，也是我们需要极力避免的：

- concurrent mode failure：并发模式失败，CMS 收集器也有同样的概念。G1 并发标记期间，如果在标记结束前，老年代被填满，G1 会放弃标记。

  > 这个时候说明
  >
  > - 堆需要增加了，
  > - 或者需要调整并发周期，如增加并发标记的线程数量，让并发标记尽快结束
  > - 或者就是更早地进行并发周期，默认是整堆内存的 45% 被占用就开始进行并发周期。

- 晋升失败：并发周期结束后，是混合垃圾回收周期，伴随着年轻代垃圾收集，进行清理老年代空间，如果这个时候清理的速度小于消耗的速度，导致老年代不够用，那么会发生晋升失败。

  > 说明混合垃圾回收需要更迅速完成垃圾收集，也就是说在混合回收阶段，每次年轻代的收集应该处理更多的老年代已标记区块。

- 疏散失败：年轻代垃圾收集的时候，如果 Survivor 和 Old 区没有足够的空间容纳所有的存活对象。这种情况肯定是非常致命的，因为基本上已经没有多少空间可以用了，这个时候会触发 Full GC 也是很合理的。

  > 最简单的就是增加堆大小

- 大对象分配失败，我们应该尽可能地不创建大对象，尤其是大于一个区块大小的那种对象。

简单小结

看完上面的 Young GC 和 Old GC 等，很多读者可能还是很懵的，这里说几句不严谨的白话文帮助读者进行理解：

首先，最好不要把上面的 Old GC 当做是一次 GC 来看，而应该当做**并发标记周期**来理解，虽然它确实会释放出一些内存。

并发标记结束后，G1 也就知道了哪些区块是最适合被回收的，那些完全空闲的区块会在这这个阶段被回收。如果这个阶段释放了足够的内存出来，其实也就可以认为结束了一次 GC。

我们假设并发标记结束了，那么下次 GC 的时候，还是会先回收年轻代，如果从年轻代中得到了足够的内存，那么结束；过了几次后，年轻代垃圾收集不能满足需要了，那么就需要利用之前并发标记的结果，选择一些活跃度最低的老年代区块进行回收。直到最后，老年代会进入下一个并发周期。

那么什么时候会启动并发标记周期呢？这个是通过参数控制的，下面马上要介绍这个参数了，此参数默认值是 45，也就是说当堆空间使用了 45% 后，G1 就会进入并发标记周期。

G1 参数配置和最佳实践

G1 调优的目标是尽量避免出现 Full GC，其实就是给老年代足够的空间，或相对更多的空间。

有以下几点我们可以进行调整的方向：

- 增加堆大小，或调整老年代和年轻代的比例，这个很好理解
- 增加并发周期的线程数量，其实就是为了加快并发周期快点结束
- 让并发周期尽早开始，这个是通过设置堆使用占比来调整的（默认 45%）
- 在混合垃圾回收周期中回收更多的老年代区块

G1 的很重要的目标是达到可控的停顿时间，所以很多的行为都以这个目标为出发点开展的。

我们通过设置 -XX:MaxGCPauseMillis=N 来指定停顿时间（单位 ms，默认 200ms），如果没有达到这个目标，G1 会通过各种方式来补救：调整年轻代和老年代的比例，调整堆大小，调整晋升的年龄阈值，调整混合垃圾回收周期中处理的老年代的区块数量等等。

当然了，调整每个参数满足了一个条件的同时往往也会引入另一个问题，比如为了降低停顿时间，我们可以减小年轻代的大小，可是这样的话就会增加年轻代垃圾收集的频率。如果我们减少混合垃圾回收周期处理的老年代区块数量，虽然可以更容易满足停顿时间要求，可是这样就会增加 Full GC 的风险等等。

下面介绍最常用也是最基础的一些参数的设置，涉及到更高级的调优参数设置，请读者自行参阅其他资料。

参数介绍：

- **-XX:+UseG1GC**

  使用 G1 收集器

- **-XX:MaxGCPauseMillis=200**

  指定目标停顿时间，默认值 200 毫秒。

  > 在设置 -XX:MaxGCPauseMillis 值的时候，不要指定为平均时间，而应该指定为满足 90% 的停顿在这个时间之内。记住，停顿时间目标是我们的目标，不是每次都一定能满足的。

- **-XX:InitiatingHeapOccupancyPercent=45**

  整堆使用达到这个比例后，触发并发 GC 周期，默认 45%。

  > 如果要降低晋升失败的话，通常可以调整这个数值，使得并发周期提前进行

- **-XX:NewRatio=n**

  老年代/年轻代，默认值 2，即 1/3 的年轻代，2/3 的老年代

  > 不要设置年轻代为固定大小，否则：
  >
  > - G1 不再需要满足我们的停顿时间目标
  > - 不能再按需扩容或缩容年轻代大小

- **-XX:SurvivorRatio=n**

Eden/Survivor，默认值 8，这个和其他分代收集器是一样的

- **-XX:MaxTenuringThreshold =n**

  从年轻代晋升到老年代的年龄阈值，也是和其他分代收集器一样的

- **-XX:ParallelGCThreads=n**

  并行收集时候的垃圾收集线程数

- **-XX:ConcGCThreads=n**

  并发标记阶段的垃圾收集线程数

  > 增加这个值可以让并发标记更快完成，如果没有指定这个值，JVM 会通过以下公式计算得到：
  >
  > ConcGCThreads=(ParallelGCThreads + 2) / 4^3

- **-XX:G1ReservePercent=n**

  堆内存的预留空间百分比，默认 10，用于降低晋升失败的风险，即默认地会将 10% 的堆内存预留下来。

- **-XX:G1HeapRegionSize=n**

  每一个 region 的大小，默认值为根据堆大小计算出来，取值 1MB~32MB，这个我们通常指定整堆大小就好了。

小结

我自己仔细检查了几遍，主要内容都囊括了，我也不知道读者看完本文会不会是一脸懵逼。

如果有什么问题，可以在留言板上给我留言，**我是 GC 的门外汉**，如果有些问题我觉得自己能解答，我会尽力帮助大家。

最后，在这里推荐一些资料给感兴趣的读者：

Oracle 官方出品，本文的很多内容是翻译并解读这篇文章的：

Getting Started with the G1 Garbage Collector

帮助大家理解 G1 的日志：