Java / Technical Details /
Technical Article

垃圾优先型垃圾回收器调优

作者：Monica Beckwith
2013 年 8 月发布

了解如何针对评估、分析和性能来调整和调优 G1 GC。

垃圾优先型垃圾回收器 (G1 GC) 是适用于 Java HotSpot VM 的低暂停、服务器风格的分代式垃圾回收器。G1 GC 使用并发和并行阶段实现其目标暂停时间，并保持良好的吞吐量。当 G1 GC 确定有必要进行垃圾回收时，它会先收集存活数据最少的区域（垃圾优先）。

垃圾回收器 (GC) 是一个内存管理工具。G1 GC 通过以下操作实现自动内存管理：

- 将对象分配给年轻代，并将老化对象晋升到老年代。
- 通过并发（并行）标记阶段，查找老年代中的存活对象。当总的 Java 堆占用率超过默认的阈值时，Java HotSpot VM 将触发标记阶段。
- 通过并行复制压缩存活对象，恢复空闲内存。

现在，我们来看看如何针对评估、分析和性能来调整和调优 G1 GC。我们假定您对 Java 垃圾回收有基本的了解。

G1 GC 是区域化、分代式垃圾回收器，这意味着 Java 对象堆（堆）被划分成大小相同的若干区域。启动时，Java 虚拟机 (JVM) 会设置区域大小。区域大小从 1 MB 到 32 MB 不等，具体取决于堆大小。目标是产生不超过 2048 个区域。Eden、存活空间和老年代是一系列不连续的逻辑区域。

G1 GC 有一个力求达到的暂停时间目标（软实时）。在年轻代回收期间，G1 GC 会调整其年轻代空间（eden 和存活空间大小）以满足软实时目标。在混合回收期间，G1 GC 会根据混合垃圾回收的目标次数调整所回收的旧区域数量，并调整堆的每个区域中存活对象的百分比，以及总体可接受的堆废物百分比。

G1 GC 将一组或多组区域（称为回收集 (CSet)）中的存活对象以增量、并行的方式复制到不同的新区域来实现压缩，从而减少堆碎片。目标是从可回收空间最多的区域开始，尽可能回收更多的堆空间，同时尽可能不超出暂停时间目标（垃圾优先）。

G1 GC 使用独立的记忆集 (RSet) 跟踪对区域的引用。独立的 RSet 可以并行、独立地回收区域，因为只需要对区域（而不是整个堆）的 RSet 进行区域引用扫描。G1 GC 使用后写屏障记录堆的更改和更新 RSet。

垃圾回收阶段

除了构成停顿 (STW) 年轻代和混合垃圾回收的疏散暂停（如下所述），G1 GC 还具有并行、并发和多阶段标记周期。G1 GC 使用初始快照 (SATB) 算法，在标记周期之初为堆中的存活对象集创建快照。存活对象集包括快照中的存活对象，以及标记周期开始以来所分配的对象。G1 GC 标记算法使用预写屏障记录和标记作为逻辑快照一部分的对象。

年轻代垃圾回收

G1 GC 可满足来自被添加到 eden 区域集的区域的大多数分配请求。在年轻代垃圾回收期间，G1 GC 会同时回收 eden 区域和上次垃圾回收的存活区域。Eden 和存活区的存活对象将被复制或疏散到新的区域集。特定对象的目标区域取决于对象的年龄；足够老的对象疏散到老年代区域（也就晋级）；否则疏散到存活区，并将包含在下一次年轻代或混合垃圾回收的 CSet 中。

混合垃圾回收

成功完成并发标记周期后，G1 GC 从执行年轻代垃圾回收切换为执行混合垃圾回收。在混合垃圾回收期间，G1 GC 可以将一些旧的区域添加到 eden 和存活区供将来回收。所添加旧区域的确切数量由一系列标志控制。关于标志的信息，将在后面讨论（请参见“掌握混合垃圾回收”）。G1 GC 回收了足够的旧区域后（经过多次混合垃圾回收），G1 将恢复执行年轻代垃圾回收，直到下一个标记周期完成。

标记周期的各个阶段

标记周期包括以下几个阶段：

- **初始标记阶段**：在此阶段，G1 GC 对根进行标记。该阶段与常规的 (STW) 年轻代垃圾回收密切相关。
- **根区域扫描阶段**：G1 GC 在初始标记的存活区扫描对老年代的引用，并标记被引用的对象。该阶段与应用程序（非 STW）同时运行，并且只有完成该阶段后，才能开始下一次 STW 年轻代垃圾回收。
- **并发标记阶段**：G1 GC 在整个堆中查找可访问的（存活的）对象。该阶段与应用程序同时运行，可以被 STW 年轻代垃圾回收中断。
- **重新标记阶段**：该阶段是 STW 回收，帮助完成标记周期。G1 GC 清空 SATB 缓冲区，跟踪未被访问的存活对象，并执行引用处理。
- **清理阶段**：在这个最后阶段，G1 GC 执行统计和 RSet 净化的 STW 操作。在统计期间，G1 GC 会识别完全空闲的区域和可供进行混合垃圾回收的区域。清理阶段在将空白区域重置并返回到空闲列表时为部分并发。

重要的默认值

G1 GC 是自适应的垃圾回收器，提供了若干默认设置，使其无需修改即可高效地工作。以下是重要选项及其默认值的列表。此列表适用于最新的 Java HotSpot VM build 24。您可以通过在 JVM 命令行输入下列选项和已更改的设置，根据您的应用程序性能需求调整和调优 G1 GC。

`-XX:G1HeapRegionSize=n`
设置的 G1 区域的大小。值是 2 的幂，范围是 1 MB 到 32 MB 之间。目标是根据最小的 Java 堆大小划分出约 2048 个区域。

`-XX:MaxGCPauseMillis=200`
为所需的最长暂停时间设置目标值。默认值是 200 毫秒。指定的值不适用于您的堆大小。

`-XX:G1NewSizePercent=5`

设置要用作年轻代大小最小值的堆百分比。默认值是 Java 堆的 5%。这是一个实验性的标志。有关示例，请参见“如何解锁实验性虚拟机标志”。此设置取代了 -XX:DefaultMinNewGenPercent 设置。Java HotSpot VM build 23 中没有此设置。

`-XX:G1MaxNewSizePercent=60`
设置要用作年轻代大小最大值的堆大小百分比。默认值是 Java 堆的 60%。这是一个实验性的标志。有关示例，请参见“如何解锁实验性虚拟机标志”。此设置取代了 -XX:DefaultMaxNewGenPercent 设置。Java HotSpot VM build 23 中没有此设置。

`-XX:ParallelGCThreads=n`
设置 STW 工作线程数的值。将 n 的值设置为逻辑处理器的数量。n 的值与逻辑处理器的数量相同，最多为 8。

如果逻辑处理器不止八个，则将 n 的值设置为逻辑处理器数的 5/8 左右。这适用于大多数情况，除非是较大的 SPARC 系统，其中 n 的值可以是逻辑处理器数的 5/16 左右。

`-XX:ConcGCThreads=n`
设置并行标记的线程数。将 n 设置为并行垃圾回收线程数 (ParallelGCThreads) 的 1/4 左右。

`-XX:InitiatingHeapOccupancyPercent=45`
设置触发标记周期的 Java 堆占用率阈值。默认占用率是整个 Java 堆的 45%。

`-XX:G1MixedGCLiveThresholdPercent=65`
为混合垃圾回收周期中要包括的旧区域设置占用率阈值。默认占用率为 65%。这是一个实验性的标志。有关示例，请参见“如何解锁实验性虚拟机标志”。此设置取代了 -XX:G1OldCSetRegionLiveThresholdPercent 设置。Java HotSpot VM build 23 中没有此设置。

`-XX:G1HeapWastePercent=10`
设置您愿意浪费的堆百分比。如果可回收百分比小于堆废物百分比，Java HotSpot VM 不会启动混合垃圾回收周期。默认值是 10%。Java HotSpot VM build 23 中没有此设置。

`-XX:G1MixedGCCountTarget=8`
设置标记周期完成后，对存活数据上限为 G1MixedGCLIveThresholdPercent 的旧区域执行混合垃圾回收的目标次数。默认值是 8 次混合垃圾回收。混合回收的目标是要控制在此目标次数以内。Java HotSpot VM build 23 中没有此设置。

`-XX:G1OldCSetRegionThresholdPercent=10`
设置混合垃圾回收期间要回收的最大旧区域数。默认值是 Java 堆的 10%。Java HotSpot VM build 23 中没有此设置。

`-XX:G1ReservePercent=10`
设置作为空闲空间的预留内存百分比，以降低目标空间溢出的风险。默认值是 10%。增加或减少百分比时，请确保对总的 Java 堆调整相同的量。Java HotSpot VM build 23 中没有此设置。

如何解锁实验性虚拟机标志

要更改实验性标志的值，必须先对其解锁。解锁方法是：在命令行中的实验性标志前，显式地设置 -XX:+UnlockExperimentalVMOptions。例如：

```
> java -XX:+UnlockExperimentalVMOptions -XX:G1NewSizePercent=10 -XX:G1MaxNewSizePercent=75 G1test.jar
```

建议

评估和微调 G1 GC 时，请记住以下建议：

- **年轻代大小**：避免使用 -Xmn 选项或 -XX:NewRatio 等其他相关选项显式设置年轻代大小。固定年轻代的大小会覆盖暂停时间目标。
- **暂停时间目标**：每当对垃圾回收进行评估或调优时，都会涉及到延迟与吞吐量的权衡。G1 GC 是增量垃圾回收器，暂停统一，同时应用程序线程的开销也更多。G1 GC 的吞吐量目标是 90% 的应用程序时间和 10%的垃圾回收时间。如果将其与 Java HotSpot VM 的吞吐量回收器相比较，目标则是 99% 的应用程序时间和 1% 的垃圾回收时间。因此，当您评估 G1 GC 的吞吐量时，暂停时间目标不要太严苛。目标太过严苛表示您愿意承受更多的垃圾回收开销，而这会直接影响到吞吐量。当您评估 G1 GC 的延迟时，请设置所需的（软）实时目标，G1 GC 会尽量满足。副作用是，吞吐量可能会受到影响。
- **掌握混合垃圾回收**：当您调优混合垃圾回收时，请尝试以下选项。有关这些选项的信息，请参见“重要的默认值”：
  - `-XX:InitiatingHeapOccupancyPercent` 用于更改标记阈值。
  - `-XX:G1MixedGCLiveThresholdPercent` 和 `-XX:G1HeapWastePercent` 当您想要更改混合垃圾回收决定时。
  - `-XX:G1MixedGCCountTarget` 和 `-XX:G1OldCSetRegionThresholdPercent` 当您想要调整旧区域的 CSet 时。

有关溢出和用尽的日志消息

当您在日志中看到目标空间溢出/用尽的消息时，意味着 G1 GC 没有足够的内存，供存活者和/或晋升对象使用。Java 堆不能扩展，因为已达到最大值。示例消息：

924.897: [GC pause (G1 Evacuation Pause) (mixed) (to-space exhausted), 0.1957310 secs]

求

要缓解此问题，请尝试以下调整：

增加 -XX:G1ReservePercent 选项的值（并相应增加总的堆大小），为“目标空间”增加预留内存量。

通过减少 `-XX:InitiatingHeapOccupancyPercent` 提前启动标记周期。

您也可以通过增加 -XX:ConcGCThreads 选项的值来增加并行标记线程的数目。

有关这些选项的描述，请参见“重要的默认值”。

巨型对象和巨型分配

对于 G1 GC，任何超过区域一半大小的对象都被视为“巨型对象”。此类对象直接被分配到老年代中的“巨型区域”。这些巨型区域是一个连续的区域集。StartsHumongous 标记该连续集的开始，ContinuesHumongous 标记它的延续。

在分配任何巨型区域之前，会检查标记阈值，如有必要，还会启动一个并发周期。

在清理阶段或完整的垃圾回收周期内，标记周期结束时会清理死亡的巨型对象。

为了减少复制开销，巨型对象未包括在疏散暂停中。完整的垃圾回收周期会对巨型对象进行压缩。

由于每个 StartsHumongous 和 ContinuesHumongous 区域集只包含一个巨型对象，所以没有使用巨型对象的终点与上个区域的终点之间的空间（即巨型对象所跨的空间）。如果对象只是略大于堆区域大小的倍数，则此类未使用的空间可能会导致堆碎片化。

如果巨型分配导致连续的并发周期，并且此类分配导致老年代碎片化，请增加 -XX:G1HeapRegionSize，这样一来，之前的巨型对象就不再是巨型对象了，而是采用常规的分配路径。

总结

G1 GC 是区域化、并行-并发、增量式垃圾回收器，相比其他 HotSpot 垃圾回收器，可提供更多可预测的暂停。增量的特性使 G1 GC 适用于更大的堆，在最坏的情况下仍能提供不错的响应。G1 GC 的自适应特性使 JVM 命令行只需要软实时暂停时间目标的最大值以及 Java 堆大小的最大值和最小值，即可开始工作。

另请参见

- 垃圾优先型垃圾回收器
- Java HotSpot 垃圾回收

关于作者

Monica Beckwith，Oracle 技术团队的主要成员，是 Java HotSpot VM 的垃圾优先型垃圾回收器的性能带头人。她在性能和架构领域已从业 10 年有余。在加入 Oracle 和 Sun Microsystems 之前，Monica 曾是 Spansion Inc. 的性能带头人。Monica 参与了许多行业级 Java 基准测试标准的制定，为 Java HotSpot 虚拟机找出改进机会是她永恒的目标。

分享交流

请在 Facebook、Twitter 和 Oracle Java 博客上加入 Java 社区对话！