[译]Java8官方GC调优指南 --(九)G1收集器 - 掘金

juejin.cn/post/6844904053617475598

2020年1月28日

> 本套文章是Java8官方GC调优指南的全文翻译，点击查看原文,原文章名称《Java Platform, Standard Edition HotSpot Virtual Machine Garbage Collection Tuning Guide》

9 Garbage-First Garbage Collector G1收集器

G1收集器是一个服务端程序的收集器，它是为带有巨量内存和多处理器的程序设计的。它尝试在满足高吞吐量的情况也能够满足停顿时间目标。所有堆相关操作，例如全局标记，都是与应用线程并发执行的。这样可以有效的防止GC停顿，效果与堆的大小或存活对象数量成正比。

G1收集器通过几项技术来达到上述目标。

堆被分割成一组大小相等的regions，每个region是一个连续的虚拟内存范围。G1执行一个并发的全局标记阶段来确定整个堆中对象的活性。标记阶段完成后，G1知道哪些rigion大部分是空的。它首先收集这些区域，这通常会产生大量的空闲空间。这就是为什么这种垃圾收集方法称为Garbage First。顾名思义，G1将收集和压缩活动集中在堆中可能充满可回收对象(即垃圾)的区域。**G1使用暂停预测模型**来满足用户定义的暂停时间目标，并根据指定的暂停时间目标选择要收集的region数。

G1将对象从堆的一个或多个region复制到堆上的另一个region，并在此过程中压缩和释放内存。这种转移是在多处理器上并行执行的，以减少暂停时间并增加吞吐量。因此，每次垃圾收集时，G1都在不断地减少碎片。这一点比前面介绍的回收器要牛逼--CMS不做压缩。Parallel只执行全堆压缩，这会导致相当长的停顿时间。

需要注意的是G1不是一个实时收集器。G1收集器能够大概率满足停顿时间的要求，但也不全是绝对的。根据之前收集的数据，G1可以提前预估在目标时间内可以收集多少个区域。因此，收集器对收集区域的成本有一个合理准确的模型，它使用这个模型来决定满足停顿时间的需求要收集哪些region和多少region。

G1的首要关注点是为用户提供一个可以在大堆的环境下保证理想GC停顿时间。这里说的堆大小约为6gb或更大，并且停顿时间低于0.5秒。

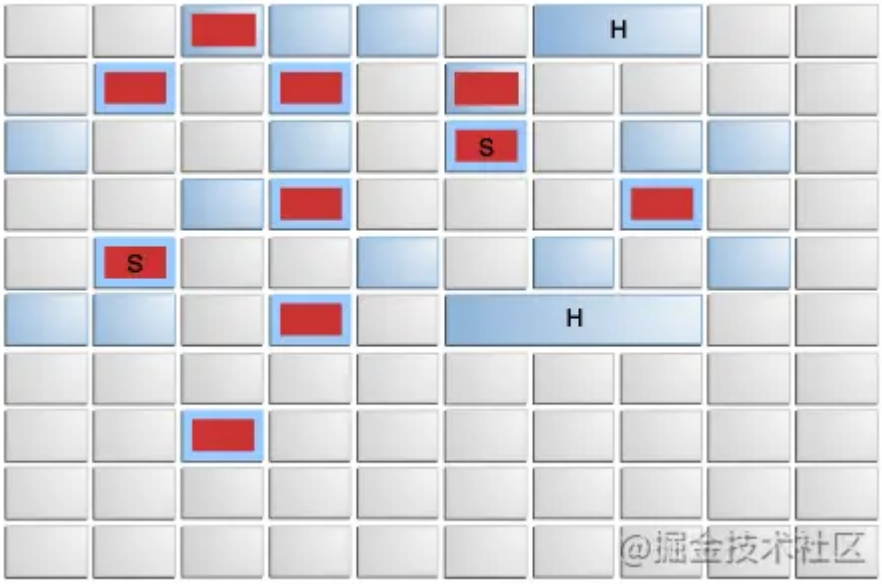
目前使用CMS或者parallel收集器的应用程序，如果你的场景满足以下几点，推荐换成G1收集器：

- Java堆有超过50%的活跃对象
- 分配对象的频率和对象升代的频率很高
- 应用程序正在经历不必要的超长停顿(停顿时间大于0.5或1秒)

G1是计划用来替代CMS的。将G1与CMS进行比较，可以发现G1要比CMS更好一些。一个不同之处在于G1是一个压缩收集器。此外，G1比CMS收集器提供了更多可预测的垃圾收集暂停，并允许用户指定预期的停顿时间。

与CMS一样，G1是为需要更短GC停顿的应用程序设计的。

如下图所示，G1将heap切分成定长的region：



> 官方认为带H标记的的宽单元格是深蓝色，带S标记的是浅蓝色。深蓝色有19个，浅蓝色有8个。

这张图包含了一个10*10的表格。大多数单元格都是灰色的。其中19个单元格是深蓝色的。这些深蓝色的单元格随机的分布在表格的上6行。其中2个单元格包含一个红方框。有两个深蓝色的单元格标记为H，比较宽。

还有8个单元格是浅蓝色的，各自包含一个红方框，其中有两个还被标记为S。这些带红方框的浅蓝色单元格随机分布在整个表格的上方。

从逻辑上说，G1也是分代的。一组空白区域被指定为logical young generation(逻辑young区)。在图中，young区是浅蓝色的。分配是在logical young generation之外完成的，当young区满时，将对该区域集进行垃圾收集(minor gc)。在某些情况下，可以同时对young区之外的区域(深蓝色代表的19个old区)进行垃圾收集。这称为mixed收集。在图中，正在收集的区域用红方框标记。该图表示当前在做mixed收集，因为同时收集了young区和tenured区。垃圾收集是一个压缩的收集，它将活动对象复制到选定的、最初为空的区域。根据幸存对象的年龄，可以将对象复制到survivor区域(用“S”标记)或复制到tenured(没显示标注出来)。标有“H”的区域内，包含了体积超过半个区域的巨型对象，并经过特殊处理;具体看Humongous Objects and Humongous Allocations in Garbage-First Garbage Collector。

Allocation (Evacuation) Failure 分配失败

与CMS一样，当应用程序运行时，G1收集器执行部分收集，可能会出现应用程序分配对象的速度比垃圾收集器恢复空间快。在G1中，当G1将活动数据从一个区域复制到另一个区域时发生故障(Java堆耗尽)。复制是为了压缩活动数据。如果在垃圾回收区域的回收过程中找不到空闲(空)区域，则会发生分配失败(因为没有空间分配)，此时会触发一次STW的停顿。

Floating Garbage 漂浮垃圾

对象可能在G1收集期间死亡但来不及收集。G1使用了一种叫做“snapshot-at-the-beginning 快照-开始”(SATB)的技术来保证垃圾收集器能够找到所有活动对象。SATB声明，在并发标记(整个堆上的标记)开始时处于活动状态的任何对象都被认为是集合活动状态的对象。这些漂浮垃圾下次GC就会被清理，和CMS相似。

Pauses 停顿

G1暂停应用程序，将活动对象复制到新区域。这些停顿可以是只收集年轻区域的young gc停顿，也可以是young区和tenured区的混合收集停顿。与CMS一样，当应用程序停止时，会有一个final mark 或 remark 停顿来完成标记阶段。CMS有initial mark pause，G1将initial mark pause作为evacuation pause的一部分。G1在收集末尾会有一个清理阶段，部分是STW，部分是并发的。G1在清理阶段的STW部分去识别空的region和tenured区域的region，为下一次收集做准备。

Card Tables and Concurrent Phases 卡牌表和并发阶段

如果垃圾收集器没有收集整个堆，那么heap对于收集器来说，是有分区的，在这种情况下，可能会存在一些指针从未收集区域指向正在被收集的区域的对象，就好比在做young gc时，young的一些对象被tenured区的对象引用，垃圾回收器必须要知道这些指针在哪里，暂且我们将这种指针叫做跨区指针，跨区指针是很烦的，因为它的存在，可能会导致young区的一些对象是无法被收集的。对于分代的垃圾收集器来说，这是非常典型的场景，其中堆中未收集的部分通常是tenured区，而堆中收集的部分是young区。用于保存这些指针的数据结构(tenured的指针指向young的对象)被称为remembered set(记忆集合)。Card Table是一种特殊类型的remembered set。JVM使用一个字节数组来表示一个Card Table。每个Byte字节对应一个Card。一个Card对应堆中的一段地址范围。Dirtying a card意味着将字节的值改为脏值;脏值可能包含一个跨区指针。

Processing a card代表查看这个card是否包含跨区指针，如果有的话，会做一些处理，例如将这个数据结构转换成其他的数据结构。

G1包含一个并发标记阶段，会标记应用中所有已发现的活跃对象。并发标记从疏散暂停结束(初始标记工作在此完成)直到remark阶段。并发清理阶段将GC后清空的region添加到free region列表中，并在remember set中清除这些区域。此外，并发细化线程根据需要去处理被应用程序写操作弄脏的、可能具有跨区指针的card table entry。

Starting a Concurrent Collection Cycle 启动一次并发收集循环

如前所述，G1使用混合收集来收集young和old区域。为了收集tenured区，G1对堆中的活动对象进行全部标记。这种标记由并发标记阶段完成。当整个Java heap的使用率达到参数 `-XX:InitiatingHeapOccupancyPercent=<NN>` 设定的值时就会触发并发标记阶段。这个参数的默认值是45。

Pause Time Goal 停顿时间目标

通过参数 `MaxGCPauseMillis` 来给G1收集器指定停顿时间目标。G1使用预测模型来估算停顿时间内可以做多少回收工作。在一次gc结束时，G1选择下次将要回收的region(collection set---GC集合)。GC集合会包括young区。在一定程度上，G1通过选择集合中young区的数量来控制GC停顿时间。与其他垃圾收集器一样，你可以通过参数指定young区的大小，但是这样做可能会影响G1达到停顿时间目标的效果。除了停顿时间目标之外，还可以指定停顿的间隔时长。 停顿的间隔时长可以通过参数 `GCPauseIntervalMillis` 指定，默认是0； `MaxGCPauseMillis` 默认是200毫秒。