深入了解标记–清扫回收算法

原创 foolishAndStupid 2017-05-20 11:06:36 2033 收藏 版权

分类专栏： jvm 垃圾回收 文章标签： 虚拟机 gc 垃圾回收

之前对标记-清扫回收算法的理解只是读完《深入理解Java虚拟机》里面的介绍，而对里面的很多细节不甚了解。看了《the garbage collection handbook》才知道里面还大有名堂。标记-清扫是怎么标记的？标记位在哪？标记过程有什么优化方法？清扫过程如何清扫？怎样优化清扫过程？回收器又是怎么和分配对象空间的分配器合作的？

回收过程分为2个阶段:

1. 追踪阶段：回收器从根集合（寄存器，线程栈，全局变量）开始遍历对象图并按照一定规则进行标记
2. 清扫阶段：回收器检查堆中每一个对象，然后将所有未标记的对象当做垃圾进行回收

标记清扫属于**间接回收**，它并非直接检测垃圾本身，而是先 确定存活对象，再反过来判断其他对象都是垃圾。**每次调用该回收算法都需要重新计算存活对象集合。**

回收器和赋值器之间的调用关系：如果线程无法再分配对象了，则唤起回收器，回收完后再次尝试分配，若分配失败，则抛出OOM异常。

回收器如何标记垃圾：回收器可以通过在对象头部某个位或者字节的方式对其进行标记，也可以标记在另外的一张表中。
标记阶段：工作列表为空的时候表示标记完成，此时表示回收器已经对每个可达对象完成的访问和标记，没有打上标记的则表明是垃圾。
清扫阶段：回收器会在堆中进行线性扫描，即从堆底开始释放未标记的对象，然后将这些对象返还给分配器，用于后面的内存分配需求。同时还会清空存活对象的标记用于下次回收时使用。

标记–清扫回收器要求堆布局满足一定的条件：

1. 该回收器不会移动对象，因此内存管理必须能控制堆内存碎片。因为过多的内存碎片会导致分配器无法满足新的分配请求，导致回收频率提高。
2. 清扫器必须能够遍历堆中的每一个对象。

时间局部性：一旦程序访问了某个地址空间，则很可能在不久之后再次访问该地址，所以值得将该值缓存
空间局部性：一旦程序访问了某个地址空间，则很可能在不久之后访问到该地址附近的地址空间。

如何改进标记–清楚算法：

位图标记

位图标记：前面提到回收器可以通过在对象头部某个位或者字节的方式对垃圾进行标记，也可以用一个独立的位图来进行标记：位图中的每个位对应着每个可能分配对象的地址。位图可以只有一个，也可以有多个，比如每个内存块维护各自的位图。
位图标记通常仅适用于单线程环境，因为多线程同时修改位图存在较大的写冲突。相对于位图，实践中更常用的是字节图（byte-map）,虽然占用的空间是前者的8倍，却可以解决写冲突问题。在实际中，如果将标记位保存在对象头部，会引起额外的复杂性。因为对象头一般存放赋值器共享的数据，如锁和哈希值。当标记线程和赋值器线程并发执行时可能会产生冲突。所以为了确保安全，一般标记位会占用对象头部额外的一个字，以便与赋值器共享的数据进行区分。

位图标记的优点：

1. **标记位更加密集**
2. **标记过程只需要读取存活对象的指针域而不**
3. **清扫器不会对存活对象进行任何读写操作，**

如将它连接到空闲链表上所以位图标记不仅可以减少内存中需要修改的字节数，而且减少了对高速缓存行的写入，进而减少需要写回内存的数据量

4.减少回收过程的内存换页次数。许多证据表明，对象往往成簇出现，成簇死亡，而许多分配器往往也会将这些对象分配在相邻空间。好处：1.在位图/字节图中，每个位或者每个字节全部都被设置/清空标记位的情况经常出现，因此回收器可以批量读取/清空一批对象的标记位；2.通过位图可以更简单的判定某一个内存块中的所有对象是否都是垃圾，进而可能一次性回收整个内存块。

混合标记策略：将每一个数据块与字节图中的一个字相关联，同时依然保留对象头部的标记位。**当且仅当内存块中至少存在一个存活对象时，该内存块所对应的标记字节才会被设置**。所以清扫器可以根据字节图快速的判断某一个内存块是否完全为空，进而整体回收。

懒惰清扫

懒惰清扫：标记过程的时间复杂度：O(L),L是堆中存活对象的数量。清扫过程中的时间复杂度是O(H),H是堆空间大小。H>L,所以我们会误认为清扫阶段的开销是整个标记-清扫开销的主要部分。但实际上，标记追踪阶段内存的访问模式是不可预测的，而清扫过程的可预测性则高的多，同时清扫一个对象的开销也比 追踪的开销小得多。

如何降低甚至消除清扫阶段的停顿时间?

基于2个特征：

1.一个对象一旦成为垃圾，它将一直是垃圾，不可能再次被赋值器访问；

2.赋值器不会访问对象的标记位。在回收过程中，回收器在将堆中所有存活对象标记完成之后，只是简单的将完全为空的内存块返还给分配器，同时将其他内存块添加到其所对应空间大小分级的回收队列中。**对于任何内存分配的请求，分配器首先从合适的空间大小分级中分配一个空闲槽，如果调用失败则调用清扫器执行懒惰清扫，即从该空间大小分级的回收队列中取出一个或者多个内存块进行清扫，直到满足分配要求。**也可能会出现没有内存块可供清扫或者是被清扫的内存块不包含任何空闲槽的情况。此时分配器就要尝试从更低级别的块分配器中获取新内存。但若无法获取新内存块，则必须执行垃圾回收

标记–清扫回收器需要考虑的问题：

1.赋值器的开销：最简单的标记-清扫回收器不会给赋值器带来任何额外的读写开销，相比之下，引用计数器会带来显著的开销。分代回收器，并发回收器，增量回收器都要求赋值器在修改指针时通知回收器，但对于程序的整体执行时间而言，这部分的开销通常较小。

2.使用懒惰清扫策略的回收器通常有较高的吞吐量，主要开销在于追踪阶段。对于已发现的垃圾对象，只需要设置一个标记位。相比之下，复制算法和标记-整理算法则还需要移动对象。

3.**空间利用率：**如果将标记位放在对象头部，则不会产生额外的开销。**如果使用位图，则额外空间取决于对象的字节要求，如要求32字节对齐，则总开销不会超过堆内存的1/32或1/64**(一个32字节的对象用一个字节来标记，所以总占用空间是所有对象大小的1/32，也就是堆内存的1/32)。而复制式算法的利用率则较低。但非整理式回收器（如标记-清理）通常都会存在内存碎片，这对空间利用率也会造成一定的影响。

和其他追踪式回收算法一样，标记-清扫算法需要在回收所有死亡对象空间之前先确定所有的存活对象。**这意味着追踪式回收器需要保留一定的空间来执行这项操作。**若堆中存活对象较多，且分配速率较快，则会引起频繁的垃圾回收操作。对于中大型的堆，需要保留20%~50%的空间。

4.移动还是不移动。非移动算法的优点在于，不移动对象的特征使得标记-清扫算法可以用于 那些编译器和回收器不合作的场景。回收器无法获取到赋值器根集合和对象域的详细信息，所以不能随便移动对象。同时，出于安全考虑，在不合作的系统中保守式的回收器不能修改用户数据，包括对象头，所以用位图进行标记要比在对象头进行标记更好。但非移动算法的主要问题在于随着回收进行会产生内存碎片，所以会比整理式算法的回收频率更高。也**需要保留20%~50%的空间，避免性能颠簸。**因为会产生内存碎片，所以许多标记-清扫算法还会定期使用标记-整理算法进行内存整理。

总结：

总结起来，标记-清扫分为标记和清扫2个过程，回收过程不会移动对象，因此容易产生内存碎片。**针对2个过程，分别有优化的方法。标记过程的优化可以采用位图（字节图）进行标记的方法。清扫过程的优化可以采用懒惰清扫的策略。**