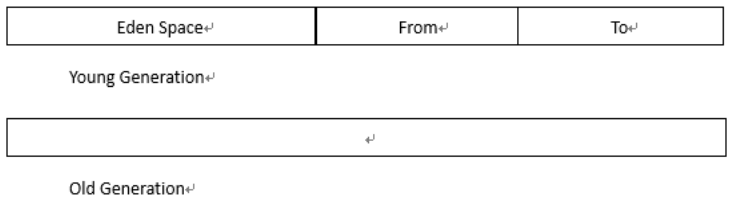
垃圾收集算法——分代收集算法（Generational Collection）。

转载 孤芳不自賞 2018-03-29 09:28:00 7052 收藏 21
分类专栏： 算法 文章标签： jvm 算法

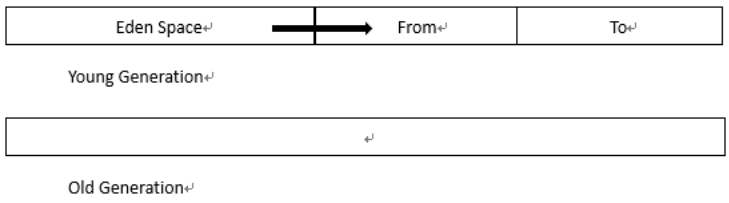
当前商业虚拟机的垃圾收集都采用“分代收集”（Generational Collection ）算法，这种算法并没有什么新的思想，只是根据对象存活周期的不同将内存划分为几块。一般是把Java堆分为新生代和老年代，这样就可以根据各个年代的特点采用最适合的收集算法。在新生代中，每次垃圾收集时都发现有大批对象死去，只有少量存活，那就选用复制算法，只需要付出少量存活对象的复制成本就可以完成。而老年代中因为对象存活率高、没有额外空间对他进行分配担保，就必须使用“标记-清理”或者“标记-整理”算法来进行回收。

在Java虚拟机分代垃圾回收机制中，应用程序可用的堆空间可以分为***年轻代***与***老年代***，然后年轻代有被分为***Eden区，From区与To区***。

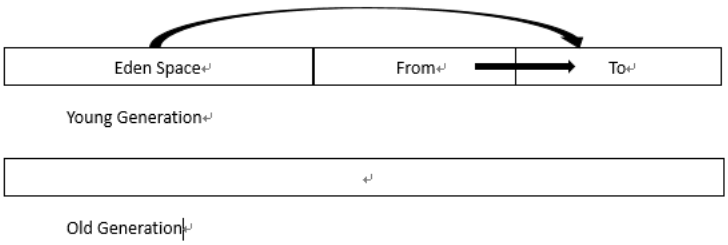


当系统创建一个对象的时候，总是在Eden区操作，当这个区满了，那么就会触发一次***YoungGC***，也就是***年轻代的垃圾回收***。

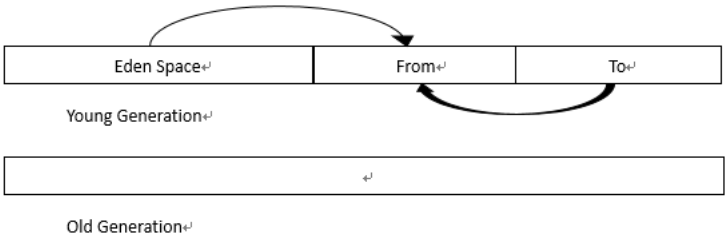
一般来说这时候不是所有的对象都没用了，所以就会把还能用的对象复制到From区。



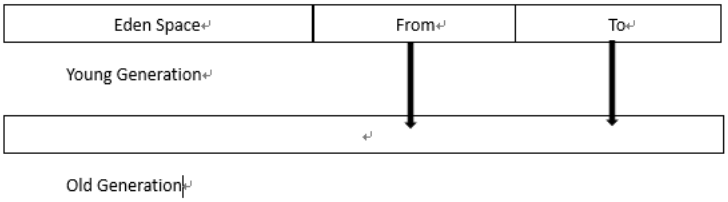
这样整个Eden区就被清理干净了，可以继续创建新的对象，当Eden区再次被用完，就再触发一次YoungGC，然后呢，注意，这个时候跟刚才稍稍有点区别。这次触发YoungGC后，***会将Eden区与From区还在被使用的对象复制到To区***，



再下一次YoungGC的时候，则是将***Eden区***.



经过若干次YoungGC后，有些对象在From与To之间来回游荡，这时候From区与To区亮出了底线（阈值），这些家伙要是到现在还没挂掉，对不起，一起滚到（复制）老年代吧。



老年代经过这么几次折腾，也就扛不住了（空间被用完），好，那就来次集体大扫除（***Full GC***），也就是全量回收，一起滚蛋吧。

全量回收呢，就好比我们刚才比作的大扫除，毕竟动做比较大，成本高，不能跟平时的小型值日（Young GC）相比，所以如果Full GC使用太频繁的话，无疑会对系统性能产生很大的影响。

所以要合理设置年轻代与老年代的大小，尽量减少Full GC的操作

参考文献

[1] 《深入理解Java虚拟机——JVM高级特征与最佳实践》，周志明，机械工业出版社。
[2] 《大型网站技术架构——核心原理与案例分析》，李智慧，电子工业出版社。

转载：https://blog.csdn.net/sinat_36246371/article/details/52998505