

深入解析Java垃圾回收机制



六尺帐篷 (/u/f8e9b1c246f1)
2017.08.05 15:28 字数 2520 阅读 1276 评论 2 喜欢 60
(/u/f8e9b1c246f1)

编辑文章 (/writer#/notebooks/15069334/notes/15429291)

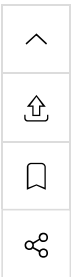
- 引入垃圾回收
- 哪些内存需要回收?
 - 引用计数法
 - 可达性分析
- 如何回收
 - Marking 标记
 - Normal Deletion 清除
 - Deletion with Compacting 压缩
- 为什么需要分代收集?
- JVM的分代
 - 新生代
 - 老年代
 - 永久代
- 分代垃圾收集过程详述

引入垃圾回收

程序计数器、虚拟机栈、本地方法栈3个区域随线程而生，随线程而灭；栈中的栈帧随着方法的进入和退出而有条不紊地执行着出栈和入栈操作。每一个栈帧中分配多少内存基本上是在类结构确定下来时就已知的（尽管在运行期会由JIT编译器进行一些优化，但在本章基于概念模型的讨论中，大体上可以认为是编译期可知的），因此这几个区域的内存分配和回收都具备确定性，在这几个区域内就不需要过多考虑回收的问题，因为方法结束或者线程结束时，内存自然就跟着回收了。而Java堆和方法区则不一样，一个接口中的多个实现类需要的内存可能不一样，一个方法中的多个分支需要的内存也可能不一样，我们只有在程序处于运行期间时才能知道会创建哪些对象，这部分内存的分配和回收都是动态的，垃圾收集器所关注的是这部分内存——《深入理解Java虚拟机》

自动垃圾回收机制就是寻找Java堆中的对象，并对对象进行分类判别，寻找出正在使用的对象和已经不会使用的对象，然后把那些不会使用的对象从堆上清除。
自动垃圾回收机制就是要解决三个问题：

- 哪些内存需要回收?
- 什么时候回收?
- 如何回收?



哪些内存需要回收？

引用计数法

对于第一个问题，也就是判断是否还需要使用，最简单的方法就是通过目前是否有引用指向这个对象，如果没有就说明这个对象不会再被使用了，如果有就说明这个对象可能还会继续被使用，这种通过引用是否存在的方法就叫做引用计数法，但这个方法存在一个问题就是无法解决对象循环引用的问题，因此又出现了可达性分析的方法来判断对象是否可以被回收。

可达性分析

这个算法的基本思路就是通过一系列的称为“GC Roots”的对象作为起始点，从这些节点开始向下搜索，搜索所走过的路径称为引用链（Reference Chain），当一个对象到GC Roots没有任何引用链相连（用图论的话来说，就是从GC Roots到这个对象不可达）时，则证明此对象是不可用的。

在Java语言中，可作为GC Roots的对象包括下面几种：

- 虚拟机栈（栈帧中的本地变量表）中引用的对象。
- 方法区中类静态属性引用的对象。
- 方法区中常量引用的对象。
- 本地方法栈中JNI（即一般说的Native方法）引用的对象。

如何回收

垃圾收集器通常会帮我们在后台自动进行垃圾回收。关于具体的回收过程只要有以下步骤

- Step 1: Marking 标记

第一步就是标记，也就是垃圾收集器会找出那些需要回收的对象所在的内存和不需要回收的对象所在的内存，并把它们标记出来，简单的说，也就是先找出垃圾在哪

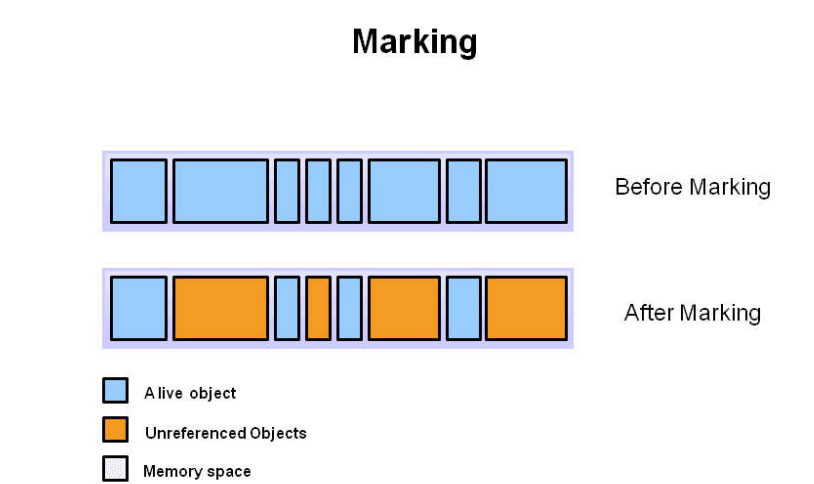
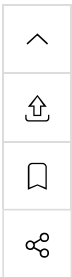


image.png



所有堆中的对象都会被扫描一遍，以此来确定回收的对象，所以这通常会是一个相对比较耗时的过程

- Step 2: Normal Deletion
垃圾收集器会清除掉上一步标记出来的那些需要回收的对象区域

Normal Deletion

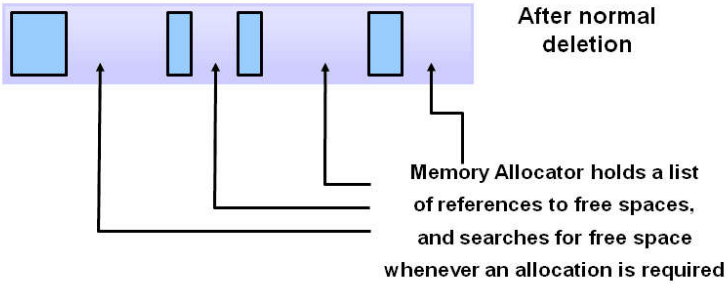


image.png

存在的问题就是碎片问题：
标记清除之后会产生大量不连续的内存碎片，空间碎片太多可能会导致以后在程序运行过程中需要分配较大对象时，无法找到足够的连续内存而不得不提前触发另一次垃圾收集动作。

- Step 2a: Deletion with Compacting 压缩
由于简单的清除可能会存在碎片的问题，所以又出现了压缩清除的方法，也就是先清除需要回收的对象，然后再对内存进行压缩操作，将内存分成可用和不可用两大部分

Deletion with Compacting

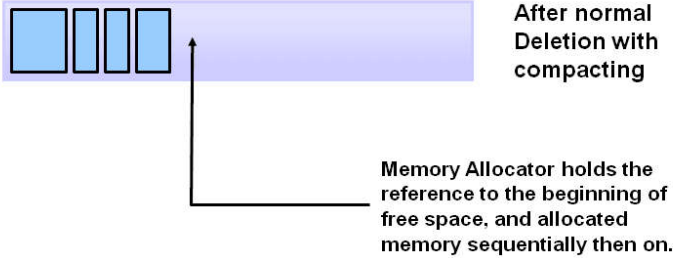
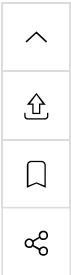


image.png



为什么需要分代收集？

就像前文所述，标记对象和压缩内存的过程在JVM中是不高效的，分配的对象越多，垃圾收集的时间就越长。但是，经过一些经验型性的统计分析表明，一个程序中大部分对象都是短命的！

下图就是一个类似的统计数据，纵坐标表示分配对象所占用的内存大小，横坐标表示自分配对象过去的时间

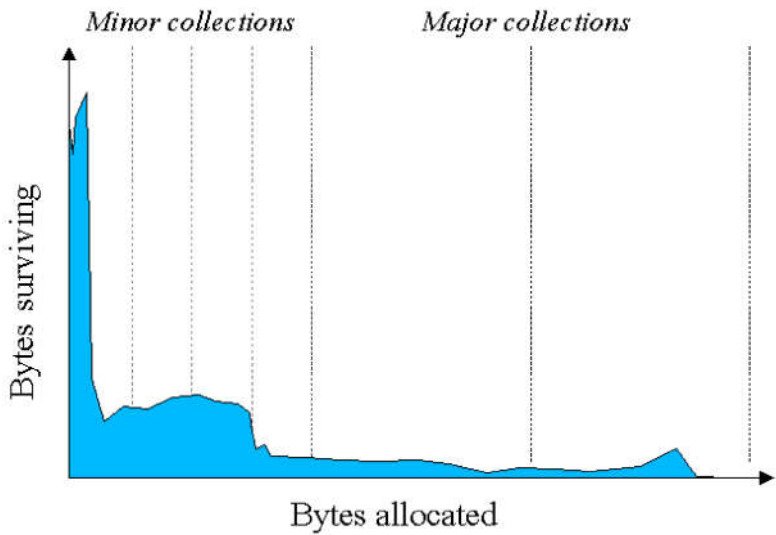
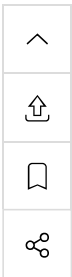


image.png

从图中我们看到，大部分对象没活多久就死了，存活较久的只是少类对象

JVM的分代

为了增大垃圾收集的效率，所以JVM将堆进行分代，分为不同的部分，一般有三部分，新生代，老年代和永久代



Hotspot Heap Structure

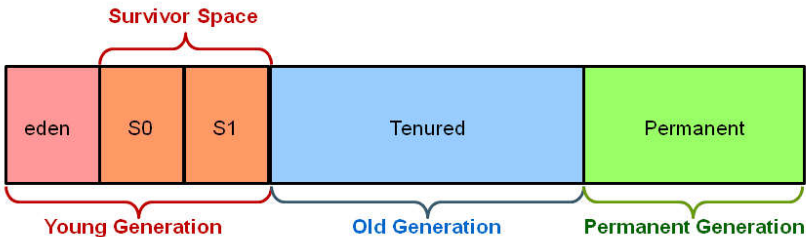


image.png

新生代

所有新new出来的对象都会最先出现在新生代中，当新生代这部分内存满了之后，就会发起一次垃圾收集事件，这种发生在新生代的垃圾收集称为Minor collections。这种收集通常比较快，因为新生代的大部分对象都是需要回收的，那些暂时无法回收的就会被移动到老年代。

Stop the World事件-所有minor garbage collections都是**Stop the World**事件，也就是意味着所有的应用线程都需要停止，直到垃圾回收的操作全部完成。类似于“你妈妈在给你打扫房间的时候，肯定也会让你老老实实地在椅子上或者房间外待着，如果她一边打扫，你一边乱扔纸屑，这房间还能打扫完？”

老年代

老年代用来存储那些存活时间较长的对象。一般来说，我们会给新生代的对象限定一个存活的时间，当达到这个时间还没有被收集的时候就会被移动到老年代中。老年代区域的垃圾收集叫做major garbage collection

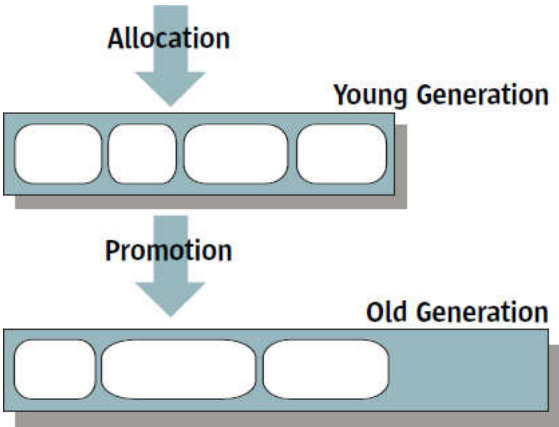


image.png

Major garbage collection也是一个**Stop the World**事件。通常Major garbage collection都相对比较慢，因为老年代的收集包括了对所有对象的收集，也就是同时需要收集新生代和老年代的对象。

^

📄

🔖

🔗

永久代

The Permanent generation contains metadata required by the JVM to describe the classes and methods used in the application. The permanent generation is populated by the JVM at runtime based on classes in use by the application. In addition, Java SE library classes and methods may be stored here.

Classes may get collected (unloaded) if the JVM finds they are no longer needed and space may be needed for other classes. The permanent generation is included in a full garbage collection.

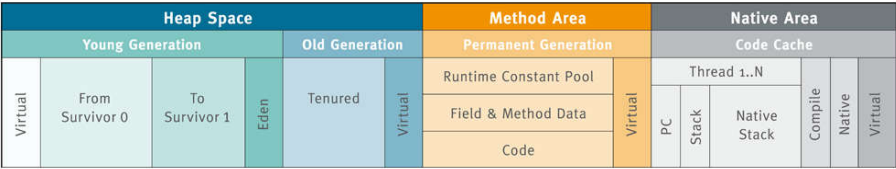


image.png

分代垃圾收集过程详述

我们已经知道垃圾回收所需要的方法和堆内存的分代，那么接下来我们就来具体看一下垃圾回收的具体过程

- 第一步 所有new出来的对象都会最先分配到新生代区域中，两个survivor区域初始化为空的

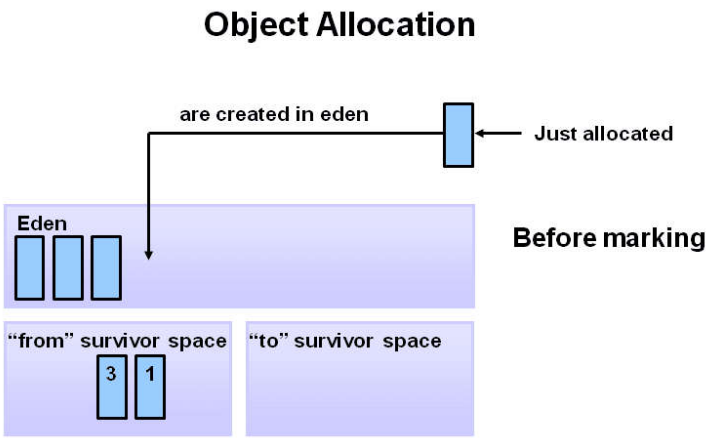


image.png

- 第二步，当eden区域满了之后，就引发一次 minor garbage collection

^

📄

🔖

🔗

Filling the Eden Space

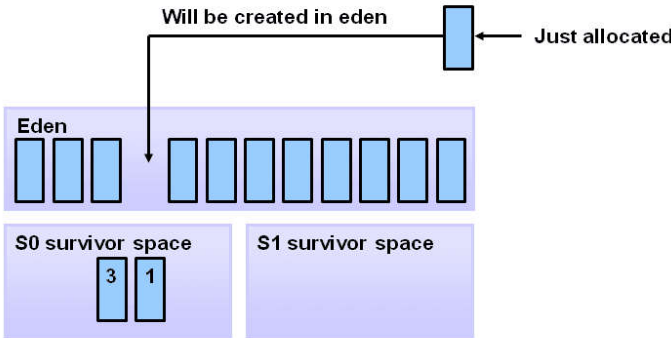


image.png

- 第三步，当在minor garbage collection，存活下来的对象就会被移动到S0survivor区域

Copying Referenced Objects

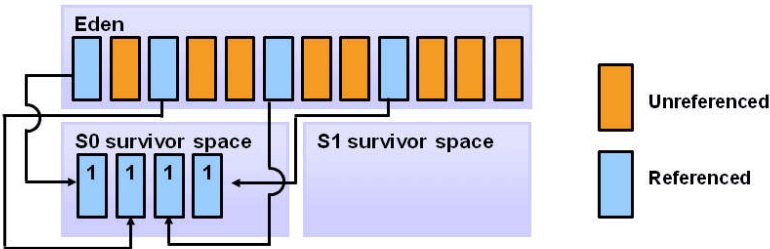


image.png

- 第四步，然后当eden区域又填满的时候，又会发生下一次的垃圾回收，存活的对象会被移动到survivor区域而未存活对象会被直接删除。但是，不同的是，在这次的垃圾回收中，存活对象和之前的survivor中的对象都会被移动到s1中。一旦所有对象都被移动到s1中，那么s2中的对象就会被清除，仔细观察图中的对象，数字表示经历的垃圾收集的次数。目前我们已经有不同的年龄对象了。

^

📄

🔖

🔗

Object Aging

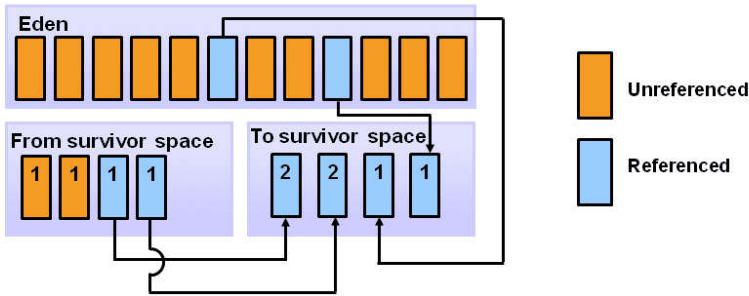


image.png

- 第五步，下一次垃圾回收的时候，又会重复上次的步骤，清除需要回收的对象，并且又切换一次survivor区域，所有存活的对象都被移动至s0。eden和s1区域被清除。

Additional Aging

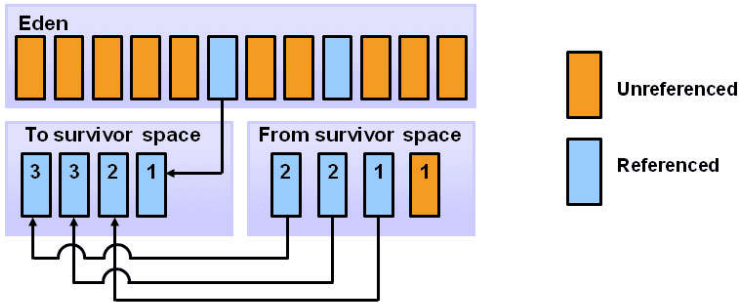


image.png

- 第六步，重复以上步骤，并记录对象的年龄，当有对象的年龄到达一定的阈值的时候，就将新生代中的对象移动到老年代中。在本例中，这个阈值为8.

^

📄

🔖

🔗

Promotion

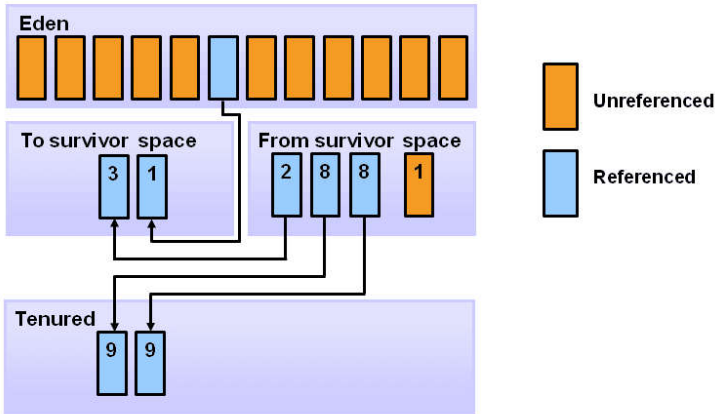


image.png

- 第七步，接下来垃圾收集器就会重复以上步骤，不断的进行对象的清除和年代的移动

Promotion

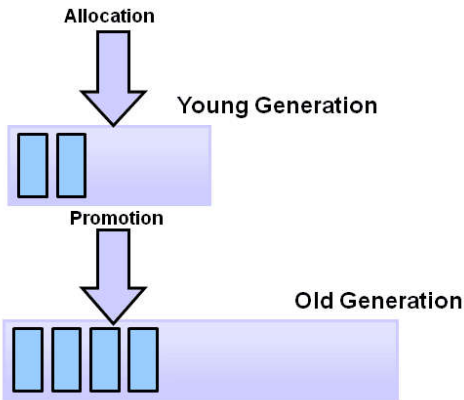


image.png

- 最后，我们观察上述过程可以发现，大部分的垃圾收集过程都是在新生代进行的，直到老年代中的内存不够用了才会发起一次 major GC，会进行标记和整理压缩。

^

📄

🔖

🔗

GC Process Summary

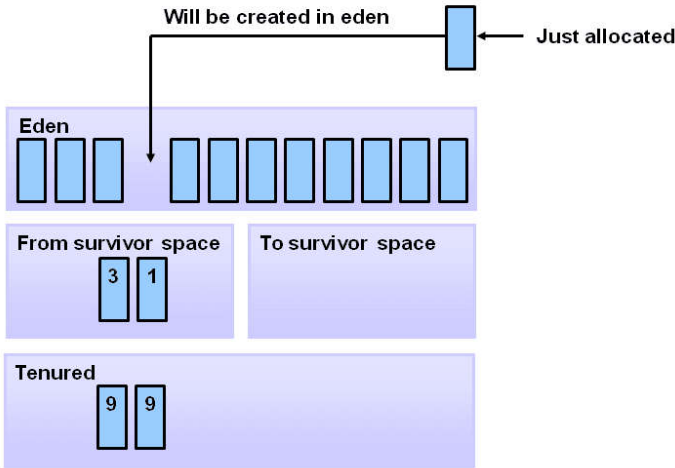


image.png

JVM (/nb/15069334)

© 著作权归作者所有



六尺帐篷 (/u/f8e9b1c246f1)

写了 248002 字, 被 15247 人关注, 获得了 1422 个喜欢 (/u/f8e9b1c246f1)


 喜欢 60

 更多分享


(<http://cwb.assets.jianshu.io/notes/images/1542929>)


被以下专题收入, 发现更多相似内容


投稿管理

- 

+ 收入我的专题
- 

Android... (/c/5139d555c94d?utm_source=desktop&utm_medium=notes-included-collection)
- 


Android开发 (/c/d1591c322c89?utm_source=desktop&utm_medium=notes-included-collection)
- 


程序员 (/c/NEt52a?utm_source=desktop&utm_medium=notes-included-collection)
- 


Android知识 (/c/3fde3b545a35?utm_source=desktop&utm_medium=notes-included-collection)
- 


首页投稿 (/c/bDHhpK?utm_source=desktop&utm_medium=notes-included-collection)
- 


Java进阶 (/c/99d627086ee7?utm_source=desktop&utm_medium=notes-included-collection)









 Android... (/c/b1fa46ec3b08?utm_source=desktop&utm_medium=notes-included-collection)

展开更多 ▾

^

📄

🔖

🔗