**[深入理解JVM(三)——垃圾收集策略详解](https://blog.csdn.net/u010425776/article/details/51189318)**

<https://blog.csdn.net/u010425776/article/details/51189318>

Java虚拟机的内存模型分为五个部分，分别是：程序计数器、Java虚拟机栈、本地方法栈、堆、方法区。

这五个区域既然是存储空间，那么为了避免Java虚拟机在运行期间内存存满的情况，就必须得有一个垃圾收集者的角色，不定期地回收一些无效内存，以保障Java虚拟机能够健康地持续运行。

这个垃圾收集者就是平常我们所说的“垃圾收集器”，那么垃圾收集器在何时清扫内存？清扫哪些数据？这就是接下来我们要解决的问题。 

程序计数器、Java虚拟机栈、本地方法栈都是线程私有的，也就是每条线程都拥有这三块区域，而且会随着线程的创建而创建，线程的结束而销毁。那么，垃圾收集器在何时清扫这三块区域的问题就解决了。

此外，Java虚拟机栈、本地方法栈中的栈帧会随着方法的开始而入栈，方法的结束而出栈，并且每个栈帧中的本地变量表都是在类被加载的时候就确定的。因此以上三个区域的垃圾收集工作具有确定性，垃圾收集器能够清楚地知道何时清扫这三块区域中的哪些数据。

然而，堆和方法区中的内存清理工作就没那么容易了。   
堆和方法区所有线程共享，并且都在JVM启动时创建，一直得运行到JVM停止时。因此它们没办法根据线程的创建而创建、线程的结束而释放。

堆中存放JVM运行期间的所有对象，虽然每个对象的内存大小在加载该对象所属类的时候就确定了，但究竟创建多少个对象只有在程序运行期间才能确定。   
方法区中存放类信息、静态成员变量、常量。类的加载是在程序运行过程中，当需要创建这个类的对象时才会加载这个类。因此，JVM究竟要加载多少个类也需要在程序运行期间确定。   
因此，堆和方法区的内存回收具有不确定性，因此垃圾收集器在回收堆和方法区内存的时候花了一些心思。 

**堆内存的回收**

**1. 如何判定哪些对象需要回收？**

在对堆进行对象回收之前，首先要判断哪些是无效对象。我们知道，一个对象不被任何对象或变量引用，那么就是无效对象，需要被回收。一般有两种判别方式：

* 引用计数法   
  每个对象都有一个计数器，当这个对象被一个变量或另一个对象引用一次，该计数器加一；若该引用失效则计数器减一。当计数器为0时，就认为该对象是无效对象。
* 可达性分析法   
  所有和GC Roots直接或间接关联的对象都是有效对象，和GC Roots没有关联的对象就是无效对象。   
  GC Roots是指：
  1. Java虚拟机栈所引用的对象(栈帧中局部变量表中引用类型的变量所引用的对象)
  2. 方法区中静态属性引用的对象
  3. 方法区中常量所引用的对象
  4. 本地方法栈所引用的对象   
     PS：注意！GC Roots并不包括堆中对象所引用的对象！这样就不会出现循环引用。

**两者对比：**   
引用计数法虽然简单，但存在一个严重的问题，它无法解决循环引用的问题。   
因此，目前主流语言均使用可达性分析方法来判断对象是否有效。

**2. 回收无效对象的过程**

当JVM筛选出失效的对象之后，并不是立即清除，而是再给对象一次重生的机会，具体过程如下：

1. 判断该对象是否覆盖了finalize()方法
   * 若已覆盖该方法，并该对象的finalize()方法还没有被执行过，那么就会将finalize()扔到F-Queue队列中；
   * 若未覆盖该方法，则直接释放对象内存。
2. 执行F-Queue队列中的finalize()方法   
   虚拟机会以较低的优先级执行这些finalize()方法们，也不会确保所有的finalize()方法都会执行结束。如果finalize()方法中出现耗时操作，虚拟机就直接停止执行，将该对象清除。
3. 对象重生或死亡   
   如果在执行finalize()方法时，将this赋给了某一个引用，那么该对象就重生了。如果没有，那么就会被垃圾收集器清除。

**注意：**   
强烈不建议使用finalize()函数进行任何操作！如果需要释放资源，请使用try-finally。   
因为finalize()不确定性大，开销大，无法保证顺利执行。

**方法区的内存回收**

我们知道，如果使用复制算法实现堆的内存回收，堆就会被分为新生代和老年代，新生代中的对象“朝生夕死”，每次垃圾回收都会清除掉大量的对象；而老年代中的对象生命较长，每次垃圾回收只有少量的对象被清除掉。

由于方法区中存放生命周期较长的类信息、常量、静态变量，因此方法区就像是堆的老年代，每次垃圾收集的只有少量的垃圾被清除掉。

方法区中主要清除两种垃圾：   
1. 废弃常量   
2. 废弃的类

**1. 如何判定废弃常量？**

清除废弃的常量和清除对象类似，只要常量池中的常量不被任何变量或对象引用，那么这些常量就会被清除掉。

**2. 如何废弃废弃的类？**

清除废弃类的条件较为苛刻：   
1. 该类的所有对象都已被清除   
2. 该类的java.lang.Class对象没有被任何对象或变量引用   
只要一个类被虚拟机加载进方法区，那么在堆中就会有一个代表该类的对象：java.lang.Class。这个对象在类被加载进方法区的时候创建，在方法区中该类被删除时清除。   
3. 加载该类的ClassLoader已经被回收

**垃圾收集算法**

现在我们知道了判定一个对象是无效对象、判定一个类是废弃类、判定一个常量是废弃常量的方法，也就是知道了垃圾收集器会清除哪些数据，那么接下来介绍如何清除这些数据。

**1. 标记-清除算法**

首先利用刚才介绍的方法判断需要清除哪些数据，并给它们做上标记；然后清除被标记的数据。

**分析：**   
这种算法标记和清除过程效率都很低，而且清除完后存在大量碎片空间，导致无法存储大对象，降低了空间利用率。

**2. 复制算法**

将内存分成两份，只将数据存储在其中一块上。当需要回收垃圾时，也是首先标记出废弃的数据，然后将有用的数据复制到另一块内存上，最后将第一块内存全部清除。

**分析：**   
这种算法避免了碎片空间，但内存被缩小了一半。   
而且每次都需要将有用的数据全部复制到另一片内存上去，效率不高。

**解决空间利用率问题：**   
在新生代中，由于大量的对象都是“朝生夕死”，也就是一次垃圾收集后只有少量对象存活，因此我们可以将内存划分成三块：Eden、Survior1、Survior2，内存大小分别是8:1:1。分配内存时，只使用Eden和一块Survior1。当发现Eden+Survior1的内存即将满时，JVM会发起一次MinorGC，清除掉废弃的对象，并将所有存活下来的对象复制到另一块Survior2中。那么，接下来就使用Survior2+Eden进行内存分配。

通过这种方式，只需要浪费10%的内存空间即可实现带有压缩功能的垃圾收集方法，避免了内存碎片的问题。

但是，当一个对象要申请内存空间时，发现Eden+Survior中剩下的空间无法放置该对象，此时需要进行Minor GC，如果MinorGC过后空闲出来的内存空间仍然无法放置该对象，那么此时就需要将对象转移到老年代中，这种方式叫做“分配担保”。

**什么是分配担保？**   
当JVM准备为一个对象分配内存空间时，发现此时Eden+Survior中空闲的区域无法装下该对象，那么就会触发MinorGC，对该区域的废弃对象进行回收。但如果MinorGC过后只有少量对象被回收，仍然无法装下新对象，那么此时需要将Eden+Survior中的所有对象都转移到老年代中，然后再将新对象存入Eden区。这个过程就是“分配担保”。

**3. 标记-整理算法**

在回收垃圾前，首先将所有废弃的对象做上标记，然后将所有未被标记的对象移到一边，最后清空另一边区域即可。

**分析：**   
它是一种老年代的垃圾收集算法。老年代中的对象一般寿命比较长，因此每次垃圾回收会有大量对象存活，因此如果选用“复制”算法，每次需要复制大量存活的对象，会导致效率很低。而且，在新生代中使用“复制”算法，当Eden+Survior中都装不下某个对象时，可以使用老年代的内存进行“分配担保”，而如果在老年代使用该算法，那么在老年代中如果出现Eden+Survior装不下某个对象时，没有其他区域给他作分配担保。因此，老年代中一般使用“标记-整理”算法。

**4. 分代收集算法**

将内存划分为老年代和新生代。老年代中存放寿命较长的对象，新生代中存放“朝生夕死”的对象。然后在不同的区域使用不同的垃圾收集算法。

**Java中引用的种类**

Java中根据生命周期的长短，将引用分为4类。

**1. 强引用**

我们平时所使用的引用就是强引用。   
A a = new A();   
也就是通过关键字new创建的对象所关联的引用就是强引用。   
只要强引用存在，该对象永远也不会被回收。 

**2. 软引用**

只有当堆即将发生OOM异常时，JVM才会回收软引用所指向的对象。   
软引用通过SoftReference类实现。   
软引用的生命周期比强引用短一些。 

**3. 弱引用**

只要垃圾收集器运行，软引用所指向的对象就会被回收。   
弱引用通过WeakReference类实现。   
弱引用的生命周期比软引用短。 

**4. 虚引用**

虚引用也叫幽灵引用，它和没有引用没有区别，无法通过虚引用访问对象的任何属性或函数。   
一个对象关联虚引用唯一的作用就是在该对象被垃圾收集器回收之前会受到一条系统通知。   
虚引用通过PhantomReference类来实现。