Universe Scientific Publishing
Computer System Networking and Telecommunications
计算机系统网络和电信，2019,1(2)
ISSN:2661-3719 (Print); 2661-3727 (Online)

JAVA's Heap Memory Generation and Garbage Collection Mechanism

Yutong CHEN　Zhengde BAO　Chenxi LI　Yawen TANG
School of Computer and Software, Jincheng College, Sichuan University, Chengdu, Sichuan, 611731

Abstract

As an efficient programming language, JAVA's virtual machine plays an important role. Its existence makes programmers do not care about complex hardware operations and various operating systems, so as to achieve an efficient performance of compiling and running everywhere. At the same time, JVM provides a powerful memory management function similar to the operating system, so that programmers do not need to release memory manually, and focus more on the program logic to improve work efficiency.

Key Words

Garbage Recycling, JAVA Virtual Machine, Memory Generation, JVM

DOI:10.18686/jsjxt.v1i2.644

JAVA 的堆内存分代及垃圾回收机制

陈禹桐　鲍正德　李晨曦　唐娅雯
四川大学锦城学院计算机与软件学院，四川，成都 611731

摘　要

JAVA 作为一门高效的编程语言，其虚拟机功不可没，它的存在使得程序员不用关心复杂的硬件操作和多样的操作系统，实现一处编译处处运行的高效表现。同时 JVM 提供了强大的类似操作系统的内存管理功能，使得程序员不用手动进行内存释放，把更多精力放在程序逻辑上，提高工作效率。

关键词

垃圾回收；JAVA 虚拟机；内存分代；JVM

1.引言

C++程序员在编程过程中需要考虑内存的释放，通过一系列指令可以对已使用的内存进行手动释放，但 JAVA 在 C++语言之上引入了垃圾回收机制[1]，这使得程序员在编写代码时可以专一心一意考虑代码而把无用对象的回收交给 JVM，解决因为程序员忘记释放而带来的内存泄漏情况。

2.堆内存的分代思想

JAVA 虚拟机内存不是无限扩展的，当我们创建一个新对象的时候便在堆内存中开辟了一个新的空间用于存放这些对象，当这些对象不在被使用时就成为了垃圾，如果这些垃圾不被清除便会造成内存溢出的严重后果。堆内存被划分为三部分，分别为年轻代、年老代、永久代[2]。虽然不这样细分也能达到清扫垃圾对象的目的，但这样做能大大提升 GC 的性能。堆内存中对象的存活时间是不同的，如果我们把所有对象放在一起，则每次 GC 时便会重复扫描某些长期存货的对象从而拉低效率，当我们同生存时间较短的对象同长期存活的对象分开后，便每次只需要扫描新创建的对象从而提升 GC 的总体性能。

2.1 年轻代

发生在年轻代中的 GC 称为 Scavenge GC 或 Minor GC，我们创建的对象会首先放在 Eden 区，当这个区的空间不足以继续分配时 JVM 便发起一次 Scavenge GC。研究表明，年轻代中 98%的对象是存活时间很短的[3]，

所以不用将年轻代划分为1:1的部分，而是8:1:1的Eden区、Survivor From区和Survivor To区（以下简称From区和To区）。其中用于为新的对象分配内存的是Eden区和From区，To区此时处于闲置状态。在GC开始时，只会对Eden区和From区进行清理，当清理完成后，Eden区的存货对象会被复制到To区，而From区的对象的移动是不确定的，这里要引入“年龄”的概念，每进行一次Minor GC后年轻代中的存活对象年龄便增加1，当其对象年龄没有达到阈值时（这里可以通过-XX:MaxTenuringThreshold来设置设定年龄阈值，默认为15岁），就从From区复制到To区，此时第一次GC过后所有的新生代存活对象全部移动到To区，From区和Eden区处于清空的状态。

2.2 年老代

年老代的GC称为Major GC或Full GC。年老代中的对象一般生命周期较长，存活几率较高，进行GC的频率也比较低，而且每次进行的清扫时间较长，这与他的GC算法有关，（下文将会进行算法介绍）。年轻的内存区域是有限的，这就是要给存活对象设置年龄阈值的意义，当这些对象的年龄到达阈值后便会从年轻代进入年老代。年老代同时也可以帮年轻代分担分配内存空间的任务，在Survivor空间中所有相同年龄的对象大小的总和大于Survivor空间的一半时，大于这些年龄的类就可以直接进入老年代。进入年老代之后这些对象的年龄会被清零，并在年老代中每经过一次Full GC存活的对象年龄也会增加1，当年龄达到阈值（20岁）时对象便被回收。当新生代中没有足够大的连续内存空间用于给年轻代分配对象，并且年老代GC之后也没有足够的连续内存用于给新对象分配空间时就会抛出内存溢出异常。

2.3 永久代与元空间

在1.7版本之前，JVM的永久代储存的时虚拟机内存中的方法，存放的是常量、静态变量、即时编译后的代码等，是被各个线程共享的内存区域。但是在JDK1.7版本便开始了对永久代部分数据的转移，例如Symbols转移到了native heap，class statics转移到了java heap，此时用于储存他们的将不再是JVM而是本地内存。永久代与元空间中进行清扫的频率极低，这里的GC主要清扫废弃的常量和类，虽然它们中GC效率极低但却是必须进行的否则会永久代内存不够用时仍然会抛出内存溢出的异常。例如：

```
public class Test2 {
    static String  st= "test";
    public static void main(String[] args) {
        List<String> list = new
ArrayList<String>();
        for (int i=0;i<
Integer.MAX_VALUE;i++){
            String str = st + st;
            st = str;
            list.add(str.intern());
        }
    }
}
```

这段代码里我们以爆炸函数生成字符串来快速占用内存并通过两个不同版本的JDK进行对比结果。java heap space的异常是在JDK1.7中运行处的结果，而JDK1.6则是PermGen space的虚拟机永久代内存溢出，由此可以看出JVM在内存方面做出的调整。

3.JAVA虚拟机是如何实现垃圾回收的

3.1 引用计数算法

引用计数法是一种原理比较直观但效率低下的垃圾回收方法（JDK1.2之前使用的算法），每个对象都对应一个引用计数器，比如在JAVA中为某个类添加一个double类型成员变量就能实现。在创建这个类之初他的计数器值为0，没当给这个对象分配一个引用时，只要对它进行引用则它的计数器就会增加1，相反的，当指向这个对象的引用（指针）离开后此对象计数器减1，任何时候计数器值为0的对象就是无用的对象，JVM就会对这部分进行回收[4]。例如：

```
Test1 t1 = new Test1();
t1=null;
```

第一行代码在内存中开辟了一个空间并用t1指向这个空间，此时这个引用的数值为1，第二行代码将这个引用赋为空值，此时t1也就不再指向Test1对象，这个对象的数值减1成为0，也就符合了可以被回收的情况。

虽然这种方法看起来比较简便，但是在大量对象存活时对所有对象进行检查，垃圾收集器会定期在所有对象的引用计数列表上遍历，一旦发现数值为 0 的计数器就进行释放，其效率并不高。并且此方法有一个致命的弱点，就是当产生循环引用的环形数据结构时，单凭计数器的数值无法完成对无用类的回收。例如：

```
public GCObject{
    public Object instance = null;
}
public class JavaGC{
public static void main(String[] args){
        GCObject GC_A = new GCObject();
        GCObject GC_B = new GCObject();
        GC_A.instance = GC_B;
GC_B.instance = GC_A;
GC_A = null;
GC_B = null;
        System.gc();
}
}
```

例子中 GC_A 和 GC_B 形成了对象的相互引用，在创建 GC_A 时有一个引用指向 A，同时在实例化 A 的时候 B 的引用也指向了 A，这样使得 A 的计数器值应为 2，同理 GC_B 的计数器值也为 2，此时当 GC_A 和 GC_B 的引用不在指向他们时他们的计数器值分别减 1，但他们的数值为依然为 1 不会被 Minor GC 清除，但 GC_A 和 GC_B 已经成为了无用的类，所以触发的是 Full GC，从这里也可以看出当今的 JVM 并不是用的这种方法。

3.2 可达性分析算法

它是一种从根开始追踪存活对象的方法，通过一系列为“GC Roots”的对象作为根节点开始展开，从根节点到搜索到的下个节点走过的路径成为引用链，如果一个对象到与根节点之间没有任何引用链，也就是说这个节点时单独存在的，则称这个对象为垃圾对象，反之，有引用链的对象被标记为活对象[5]。这种通过引用链的追寻方法就解决了上述例子中因互相引用而产生的计数器不会为零的情况。要注意的是可达性分析算法于引用计数算法的一个很大不同在于它是一种“stop-the-world”的算法，它需要停止所有应用线程来进行遍历的算法。

C、RUBY 等运行在解释器上的语言没有进行底层变量管理，当变量不需要时，程序员要通过手动 delete 或 free 释放资源。JVM 会提供类似操作系统功能的一个逻辑连续的内存，所以如何管理内存碎片是个严重的问题。“stop-the-world”的垃圾回收模式是指在垃圾回收时需要停下全部工作，mutator 时最主要的线程。标记好的对象被移动到另一个内存中，然后清除掉无用的对象，且复制后的新内存区域会变成连续的内存区域，同时解决了内存不连续的问题。但是当内存区域移动时，他们的索引也将会改变，但这些索引会被根算法中保存，这是一个浩大的工程，这也就是要停下所有应用线程的原因，从这个角度来说这种算法的效率依然不是很高。

3.3 标记-清除算法

标记-清除算法在垃圾对象较少的时候的效果明显，所以多运用于年老代。它分为两个阶段，第一阶段只进行标记，通过引用链找到存活的对象并把他们标记出来，剩余的就是无用的对象，也就是要被清除的对象[6]。这样的结果也会导致可用的内存不连续，当我们需要分配空间给一个较大的类的时候可能出现无法找到足够内存而不得不出发另一次垃圾清扫活动。

3.4 复制算法

此方法运用与年轻代，且适用于垃圾较多的情况，它将内存划分为 1:1 的两块，每次创建新的类时只在其中一块创建，当这一块内存用完了，便把存活的类复制到另一块内存中，然后一次性把最初的内存区域全部清空。值得一提的时 JVM 有一种“自我调节”的方式，会根据垃圾的多少自动切换标记-清除算法和复制算法。

3.5 标记-整理（标记-压缩）算法

标记-清除算法适用于年老代中大多数情况，但当出现年老代中有大量存活率高的对象时就需要有额外的空间进行担保，标记-整理算法由此诞生。同标记-清除算法相似，需要先对存活的对象进行标记，但不会立即清理，而是把他们移动到内存的一端形成连续的内存区域，然后在对边界以外的内存直接清理。虽然在移动内存的过程中会耗费较多时间，但是在内存管理和长远角度来看，这是非常必要的。

3.6 分代收集算法

这个算法只是对前面的几种算法进行了分类，结合各个算法的特点和各个内存区域的对象的存活现象进行搭配，从而达到较高可能对 GC 性能进行优化的作用。

比如在存活率较低的年轻代适合使用复制算法，而在存活率较高的年老代则应该使用标记-整理算法。

4.总结

JAVA 的垃圾回收机制为 JAVA 编程人员带来了极大的简便，自动回收垃圾对象在提高了编程效率的同时防止了由于人员操作失误而引起的虚拟机崩溃的严重问题。

参考文献

[1] 潘春花.Java 与 C++垃圾回收机制剖析[J].信息系统工程,2015(12):12-13.
[2] 杨毅.Java 垃圾回收机制研究[J].计算机光盘软件与应用,2013,16(22):120-121.
[3] 张卫.综述 java 运行中垃圾回收机制[J].数字技术与应用,2017(02):231.
[4] 骆凡. Java 虚拟机中冷对象的研究[D].武汉邮电科学研究院,2016.
[5] 赫明正. 面向多处理器的并行垃圾回收机制的研究[D].哈尔滨工业大学,2011.
[6] 吕爱民,何钦铭.Java 虚拟机垃圾收集器的性能分析与调节[J].计算机应用与软件,2004(10):110-112.

作者简介

第一作者：陈禹桐（1997-），男，汉，四川省成都市，本科，四川大学锦城学院，研究方向：J2EE。
第二作者（通讯作者）：鲍正德（1989 年—），男，汉族，黑龙江哈尔滨市人，研究生，讲师，四川大学锦城学院，研究方向:电子商务
第三作者：李晨曦（1998-），男，汉，贵州省贵阳市，本科，四川大学锦城学院，研究方向：大数据技术开发