Atitit gc 内存垃圾管理算法 atgc

Catalog

[2、应用程序对资源（内存使用）管理的方式，常见的一般有如下几种： 1](#_Toc24889)

[[1] 手动管理：C,C++ 1](#_Toc6025)

[[2] 计数管理：COM 1](#_Toc9600)

[[3] 自动管理：.NET,Java,PHP,GO… 1](#_Toc6235)

[[1] 程序员忘记去释放内存 1](#_Toc6966)

[[2] 应用程序访问已经释放的内存 1](#_Toc11083)

[#自动管理：.NET,Java,PHP,GO… 2](#_Toc28746)

[1、标记 -清除算法（Mark-Sweep） 基础的收集算法 2](#_Toc17413)

[2、复制算法（Copying） 2](#_Toc6497)

[3、标记-整理(Mark-Compact) 组合（标记并且移动） 2](#_Toc15766)

[4、分代收集算法（Generational Collection） 2](#_Toc25160)

[按照对象大小分类小对象堆（small object heap，SOH）和大对象堆 3](#_Toc1451)

[时间算法 3](#_Toc22218)

[#七、当GC遇到多线程 3](#_Toc8839)

# 2、应用程序对资源（内存使用）管理的方式，常见的一般有如下几种：

## [1] 手动管理：C,C++

## [2] 计数管理：COM

## [3] 自动管理：.NET,Java,PHP,GO…

　　3、但是，手动管理和计数管理的复杂性很容易产生以下典型问题：

[1] 程序员忘记去释放内存

[2] 应用程序访问已经释放的内存

# #自动管理：.NET,Java,PHP,GO…

## **1、标记 -清除算法（Mark-Sweep）** 基础的收集算法

“标记-清除”算法，如它的名字一样，算法分为“标记”和“清除”两个阶段：首先标记出所有需要回收的对象，在标记完成后统一回收掉所有被标记的对象。之所以说它是最基础的收集算法，是因为后续的收集算法都是基于这种思路并对其缺点进行改进而得到的。

## 2、复制算法（Copying）

    为解决效率问题，“复制”收集算法出现了。它将可用内存按容量划分为大小相等的两块，每次只使用其中的一块。当这一块的内存用完了，就将还存活着的对象复制到另外一块上面，然后再把已使用过的内存空间一次清理掉。

    这样使得每次都是对其中的一块进行内存回收，内存分配时也就不用考虑内存碎片等复杂情况，只要移动堆顶指针，按顺序分配内存即可，实现简单，运行高效。

————————————————

## 3、标记-整理(Mark-Compact) 组合（标记并且移动）

    复制收集算法在对象存活率较高时就要执行较多的复制操作，效率将会变低。更关键的是，如果不想浪费50%的空间，就需要有额外的空间进行分配担保，以应对被使用的内存中所有对象都100%存活的极端情况，所以在老年代一般不能直接选用这种算法。

    根据老年代的特点，有人提出了另外一种“标记-整理”（Mark-Compact）算法，标记过程仍然与“标记-清除”算法一样，但后续步骤不是直接对可回收对象进行清理，而是让所有存活的对象都向一端移动，然后直接清理掉端边界以外的内存。

————————————————

## 4、分代收集算法（Generational Collection）

    GC分代的基本假设：绝大部分对象的生命周期都非常短暂，存活时间短。

   “分代收集”算法，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当的收集算法。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收集。而老年代中因为对象存活率高、没有额外空间对它进行分配担保，就必须使用“标记-清理”或“标记-整理”算法来进行回收。

五、GC中的代

　　代（Generation)引入的原因主要是为了提高性能（Performance)，以避免收集整个堆（Heap）。一个基于代的垃圾回收器做出了如下几点假设：

　　1、对象越新，生存期越短；

　　2、对象越老，生存期越长；

　　3、回收堆的一部分，速度快于回收整个堆。

## 按照对象大小分类小对象堆（small object heap，SOH）和大对象堆

.NET的垃圾收集器将对象分为三代（Generation0,Generation1,Generation2）。不同的代里面的内容如下：

　　1、G0 小对象(Size<85000Byte)：新分配的小于85000字节的对象。

　　2、G1:在GC中幸存下来的G0对象

　　3、G2:大对象(Size>=85000Byte);在GC中幸存下来的G1对象

CLR会管理两种类型的堆，小对象堆（small object heap，SOH）和大对象堆（large object heap，LOH），其中所有大于85000byte的内存分配都会在LOH上进行。

## 时间算法

# #七、当GC遇到多线程

　　前面讨论的垃圾回收算法有一个很大的前提就是：只在一个线程运行。而在现实开发中，经常会出现多个线程同时访问托管堆的情况，或至少会有多个线程同时操作堆中的对象。一个线程引发垃圾回收时，其它线程绝对不能访问任何线程，因为垃圾回收器可能移动这些对象，更改它们的内存位置。CLR想要进行垃圾回收时，会立即挂起执行托管代码中的所有线程，正在执行非托管代码的线程不会挂起。然后，CLR检查每个线程的指令指针，判断线程指向到哪里。接着，指令指针与JIT生成的表进行比较，判断线程正在执行什么代码。