垃圾收集器

15/7/26 19:04

程序计数器、Java虚拟机栈、本地方法栈分配的内存是确定的，生命周期与线程相同，所以不需要过多考虑回收问题。而Java堆和方法区只有运行时才知道有哪些对象被创建，需要多少内存，这部分的内存分配和回收是动态的。

1. 检测垃圾内存的方法

- 引用计数器
  给对象添加引用计数器，有地方引用时+1，引用失效时-1，任何时刻计数器为0的对象就是不可能在被使用的。但是！不能解决对象间互相引用的问题，所以主流虚拟机不用这个方法。

- 可达性分析算法
  通过一系列称为“GC Roots”的对象作为起始点，开始向下搜索，走过的路径称为引用链，当一个对象到GC Roots没有任何引用链相连时，则该对象不可用。
  可作为GC Roots的对象包括：
  - 虚拟机栈中引用的对象
  - 方法区中类静态属性应用的对象
  - 方法区中常量引用的对象
  - 本地方法栈中JNI引用的对象

2. 对象死亡过程

至少要经历两次标记过程：

- 可达性分析后不可达的对象被第一次标记并且进行一次筛选，筛选条件是对象是否有必要执行finalize()方法。当对象没有覆盖finalize()方法或者已经调用过，则没有必要执行，对象会被放进“即将回收”集合；有必要执行的对象会被放在一个叫F-Queue的队列中，会由自动建立的低优先级的Finalizer线程去触发，但不保证能运行结束。
- 在finalize()方法中对象将自己与引用链上的任何一个对象关联起来，则GC在F-Queue中进行第二次小规模标记时，这些对象会被移出回收集合，所以执行finalize()的对象不一定会被回收。任何对象的finalize()方法只能被调用一次，所以第一次逃脱后第二次将无法逃脱。

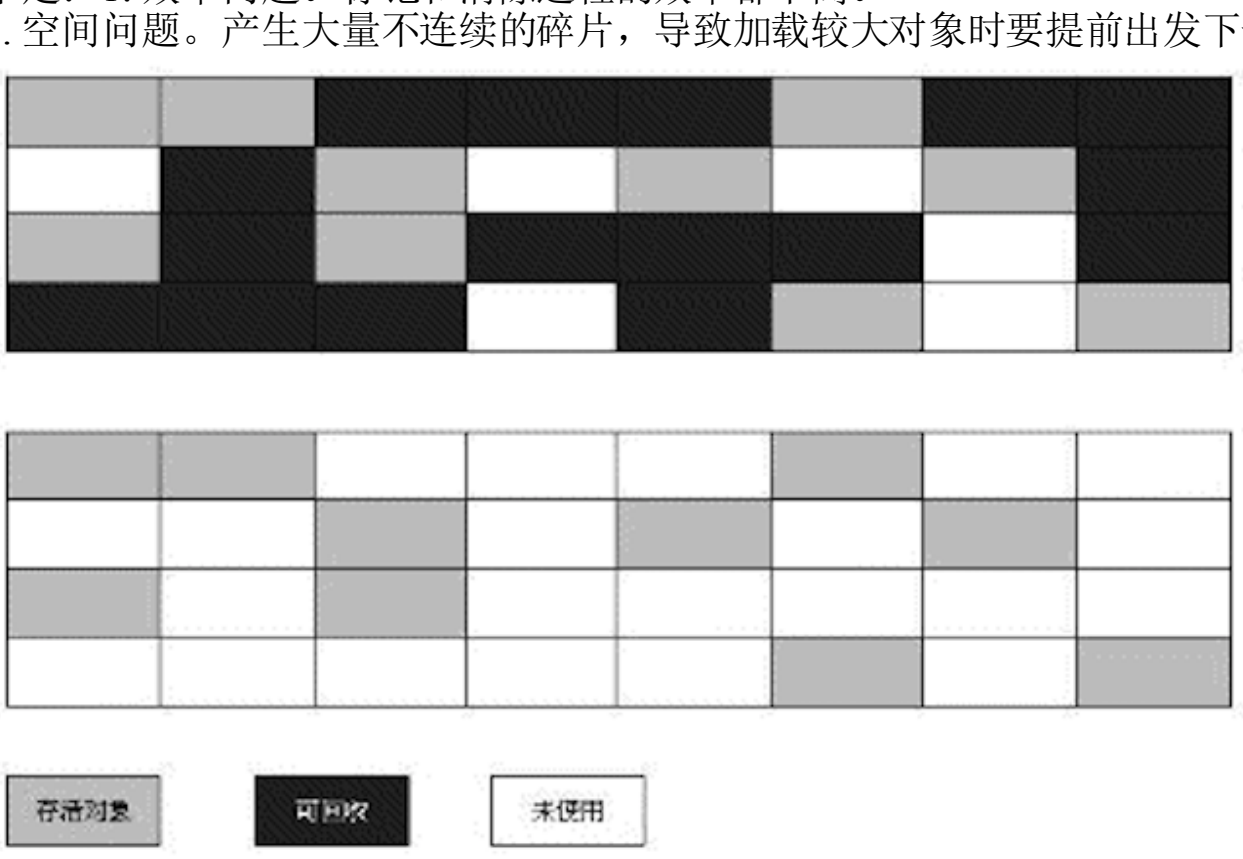
3. 回收方法区

主要回收废弃常量和无用的类。

- 废弃常量：没有其他地方被引用到的常量。
- 无用的类：满足3个条件即可以回收而并不一定：
  i. 该类的所有实例都被回收
  ii. 加载该类的ClassLoader已经被回收
  iii. 该类对应的Class对象没有在任何地方被引用

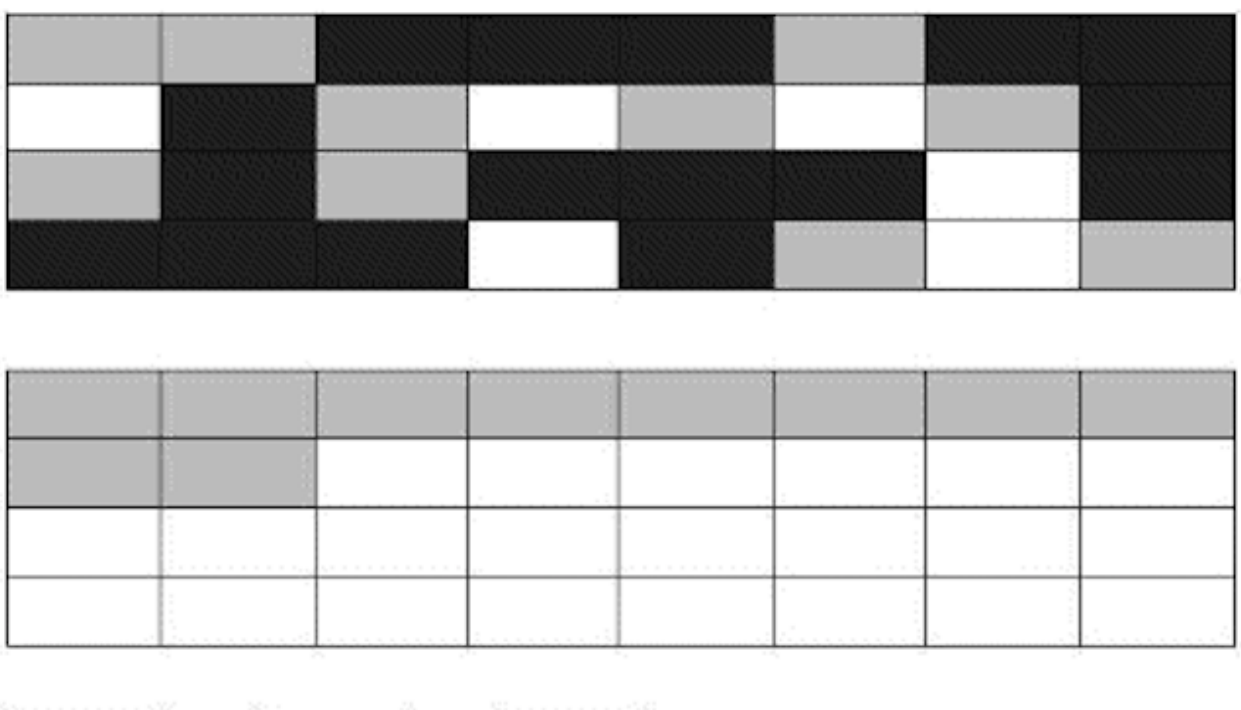
4. 垃圾收集算法

- “标记-清除”算法：
  不足：1. 效率问题。标记和清除过程的效率都不高。
  2. 空间问题。产生大量不连续的碎片，导致加载较大对象时要提前出发下一次垃圾收集过程。



- 复制算法：
  将内存分为相等大小的两块，每次只使用其中一块，当一块用完时将活着的对象复制到另一块上面，然后一次清除使用过的那块内存。
  优点：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。
  缺点：内存缩小为原来的一半，在对象存活率高的时候不适用，适合新生代。
  IBM策略：采用一个较大的Eden空间（80%）和两个较小的Survivor空间（10%），每次使用Eden和一个Survivor。

- “标记-整理”算法：
  适用于老年代，标记后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。



- 分代收集算法：
  将Java堆分为新生代和老年代，根据各个年代的特点采用最适合的收集算法。

5. 内存分配与回收策略

- 对象优先在新生代的Eden分配
- 大对象直接进入老年代（很长的字符串、数组....）
- 长期存活的对象进入老年代。每个对象都有一个年龄计数器，在Eden中出生并经历第一次GC，存活后能被Survivor容纳，则年龄置为1，每在Survivor中熬过一次GC，年龄+1，年龄增大到一定程度（默认15）则会被晋升到老年代。