**07 GC相关**

**本章理论性较强，基本上没有实操。面试也不会有GC相关的上机操作**

对象被判定为垃圾的**标准**——**没有被其他对象引用**

**判定对象是垃圾（不被引用）的算法：**

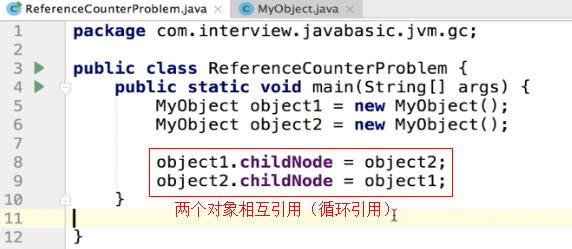
1. 引用计数算法——堆中每个对象实例都有一个引用计数器，被引用则+1，完成引用（引用到达生命周期或指向其他实例）则-1。当计数器为0时，对象实例被回收。

优点：因为只需过滤计数为0的实例，执行效率高，几乎不打断程序运行

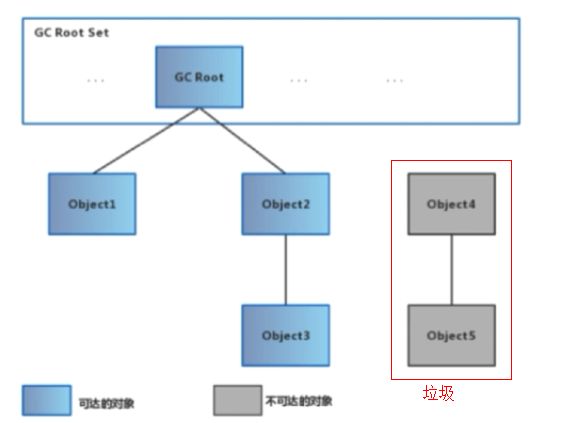
缺点：程序实现过于简单，无法检测循环引用（父对象子对象互相引用），导致内存泄漏

循环引用示例：





1. 可达性分析算法—— 判断对象的引用链是否可达决定是否回收



可用作GC Root的对象：

<1>Java虚拟机栈中引用的对象（栈帧中的本地变量表）

<2>方法区中的常量引用的对象

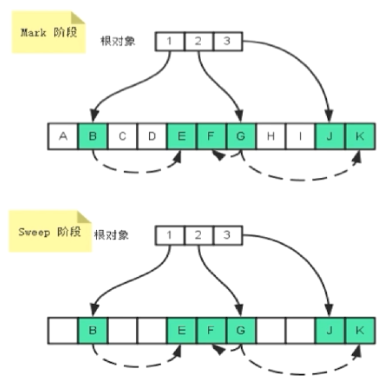
<3>方法区中的类静态属性引用的对象

<4>本地方法栈中JNI（Native方法）的引用对象

<5>活跃线程的引用对象

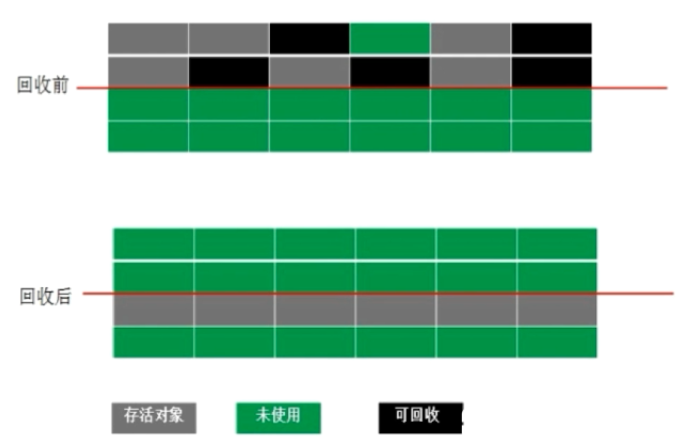
**垃圾回收算法：**

1. 标记-清除算法（Mark and Sweep）——有内存碎片化问题
2. 标记阶段：（使用可达性分析算法）从GC Root集合进行扫描，对存活的对象进行标记
3. 清除阶段：对堆内存从头到尾线性遍历，回收不可达对象的内存

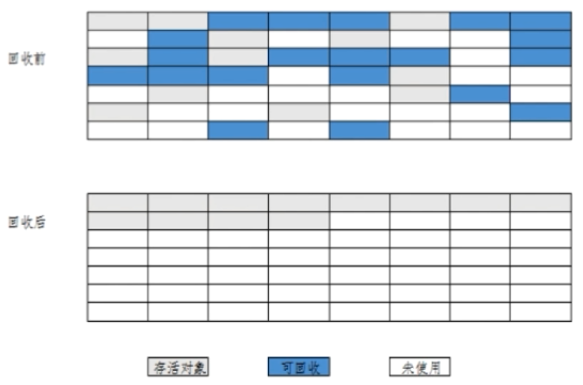


缺点：易造成碎片化

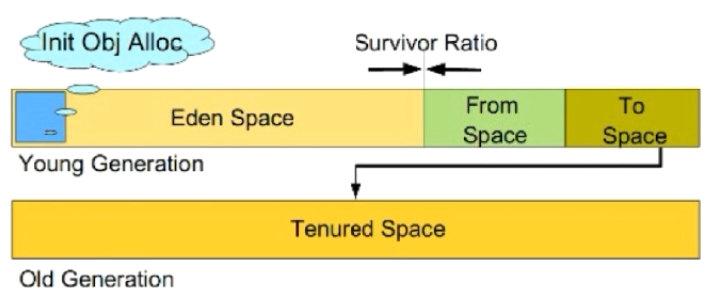
1. 复制算法（Copying）——适用于对象存活率低场景（如：新生代），无碎片化问题
2. 可用内存划分为若干块，将这些块分为对象面和空闲面
3. 对象在对象面创建
4. 对象面耗尽时，将非垃圾对象复制到空闲面
5. 将对象面清除



1. 标记-整理算法（Compacting）——适用于对象存活率高场景（如：老年代）
2. 标记阶段：（使用可达性分析算法）从GC Root集合进行搜索，对存活的对象进行标记
3. 清除阶段：移动所有存活的对象，按内存地址次序依次排列，然后将末端内存地址以后的内存全部回收

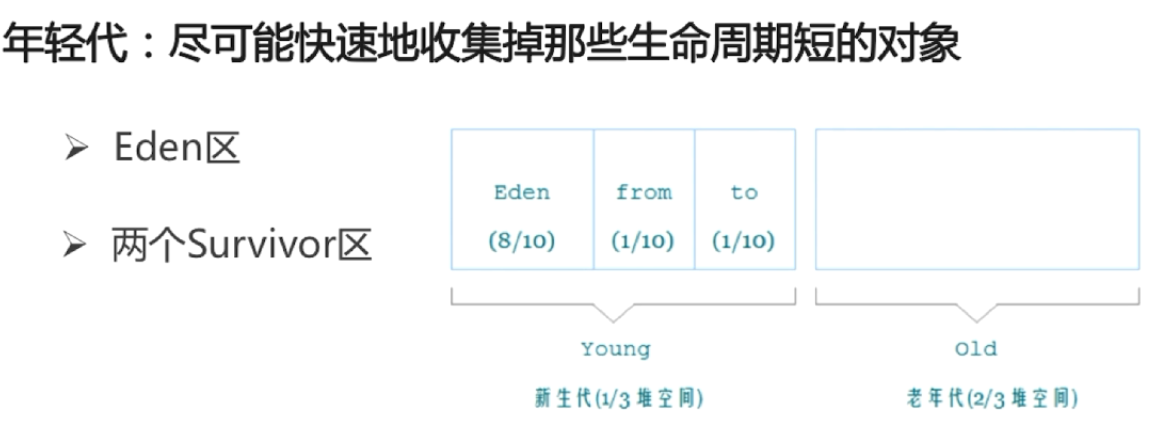


1. 分代收集算法（Generational Collector）——垃圾回收算法的组合拳。将不同生命周期的对象归到不同堆内存区域，进而采用不同回收算法。提高JVM的GC效率。**年轻代使用复制算法，老年代使用标记-清除或标记-整理算法**



堆中年轻代、老年代空间划分

分代收集算法的GC分为两种：1.Minor GC（发生在年轻代的GC，采用复制算法）；Full GC（与老年代GC相关，同时还伴随着年轻代GC）



**年轻代对象Minor-GC详细过程见“剑指Java 7-2 Java垃圾回收之回收算法”**

年轻代对象如何晋升到老年代？

答：

1. 经历一定次数Minor-GC后依然存活的对象
2. Survivor区存放不下的对象
3. 新生成的大对象（-XX：+PretenuerSizeThreshold）

**一些调优参数见“剑指Java 7-2 Java垃圾回收之回收算法”（可能不太重要）**

触发Full GC的条件：

1. 老年代空间不足
2. Minor GC晋升到老年代的平均空间大于老年代剩余空间
3. 程序员调用System.gc()

**新生代、老年代垃圾回收器介绍见“剑指Java 7-3 Java垃圾回收之新生代垃圾收集器”及“剑指Java 7-4 Java垃圾回收之老年代垃圾收集器”（个人觉得不重要）**