判断Heap中对象的生存状态

如何判断

引用计数算法（Reference Counting）

给对象中添加一个引用计数器，每当有一个地方引用它时，计数器+1；
当引用失效时，计数器-1；任何时刻计数器为0即表示对象不可能再被使用。

Java虚拟机不采用此法，最主要是因为：**它很难解决对象之间相互循环引用的问题**

可达性分析算法（Reachability Analysis）

主流商品程序语言的主流实现，包括Java，C#，Lisp

基本思路：通过一系列GC-Roots的对象作为起始点，从这些节点开始向下搜索，搜索所走过的路径称为引用链（Reference-Chain），当一个对象到GC-Roots没有任何引用链相连时，则证明此对象不可用

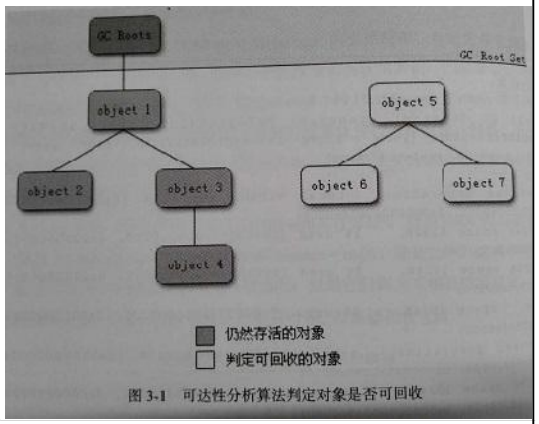

图 3-1 可达性分析算法判定对象是否可回收

GC-Roots对象包括

- 虚拟机栈（栈桢中本地变量表）中引用的对象
- 方法区中类静态属性引用的对象
- 方法区中常量引用的对象
- 本地方法栈中JNI（即Native方法）引用的对象

引用类型

- 强引用（Strong Reference）：只要引用存在，对象永远不会被GC回收，如：Object obj = new Object()
- 软引用（Soft Reference）：有用但非必需的对象，系统发生内在溢出异常之前，会把这些对象列进回收范围之中进行二次回收，回收后还没有足够内存，才会抛出内存溢出异常
- 弱引用（Weak Reference）：只能生存到下一次垃圾收集之前
- 虚引用（Phantom Reference）：唯一目的：对象被收集器回收时收到一个系统通知

finalize()方法

- 生存或死亡的最后一道防线
- 只会被系统执行一次
- 运行代价高昂，不确定性大，无法保证各个对象的调用顺序

回收方法区（永久代）

- 永久代中进行垃圾回收的'性价比'比较低
- '无用的类'满足的条件
  - 类的所有实例都已经被回收，即Heap中不存在该类的任何实例
  - 加载该类的ClassLoader已经被回收
  - 该类对应的java.lang.Class对象没有在任何地方被引用，无法在任何地方通过反射访问该类的方法
- HotSpot虚拟机控制参数
  - -Xnoclassgc
  - -verbose:class
  - -XX:+TraceClassLoading
  - -XX:+TraceClassUnLoading
- 大量使用反射，动态代理，CGLib等ByteCode框架，动态生成JSP以及OSGi这类频繁自定义ClassLoader的场景都需要虚拟机具备类卸载功能，保证永久代不会溢出