判断对象是否存活的算法包括：

* 引用计数算法
* 可达性分析算法

答：通过引用计数算法或者可达性分析算法去判断一个对象是否存活。

### 1、引用计数算法

给对象中添加一个引用计数器，每当有一个地方引用它时，计数器加1；当引用失效时，计数器值减1；任何时刻计数器为0的对象就是不能再被引用的。

例如Object-C，Python语言使用引用计数算法进行内存管理。Java虚拟机没有选用引用计数器算法来管理内存，其中最主要的原因是它很难解决对象之间相互循环引用的问题。

对象循环引用代码示例：

public class ReferenceCountingGC {

public Object instance = null;

public static void testGC() {

ReferenceCountingGC objA = new ReferenceCountingGC();

ReferenceCountingGC objB = new ReferenceCountingGC();

objA.instance = objB;

objB.instance = objA;

objA = null;

objB = null;

// 假设在这行发生GC， objA 和 objB是否能被回收？

System.gc();

}

}

对象objA和objB都有instance字段，赋值令objA.instance = objB及objB.instance = objA，初此之外，这两个对象再无任何引用，实际上这两个对象已经不能再被访问，但是它们因为相互引用着对方，导致它们的引用计数都不为0，于是引用计数算法无法通知GC收集器回收它们。



### 2、可达性分析算法

可达性分析算法的基本思路是通过一系列的称为“GC Roots”的对象作为起始点，从这些节点开始向下搜索，搜索走过的路径称为引用链（Reference Chain），当一个对象到GC Root没有任何引用链相连时，则证明此对象是不可用的。

#### 2.1 生存还是死亡

即使在可达性分析算法中不可达的对象，也并非是“非死不可”的，这时候它们暂时处于“缓刑”阶段，要真正宣告一个对象死亡，至少要经历两次标记过程：如果对象在进行可达性分析后发现没有与GC Roots相连接的引用，那它将会被第一次标记并且进行一次筛选，筛选的条件是此对象是否是否有必要执行finalize()方法。当对象没有覆盖finalize()方法，或者finalize()方法已经被虚拟机调用过，虚拟机将这两种情况都视为“没有必要执行”。

如果这个对象被判定为有必要执行finalize()方法，那么这个对象将会放置在一个叫做F-Queue的队列之中。并在稍后由一个虚拟机自动建立的，低优先级的Finalizer线程去执行它。这里所谓“执行”是指虚拟机会触发这个方法，但并不承诺会等待它运行结束，这样做的原因是，如果有一个对象在finalize()方法中执行缓慢，或者发生死循环，将可能会导致F-Queue队列中其他对象永久处于等待，甚至导致整个内存回收系统崩溃。

finalize()方法是对象逃脱死亡命运的最后一次机会，稍后GC将对F-Queue中的对象进行第二次小规模的标记，如果对象这个时候，未被重新引用，那它基本上就真的被回收了。

#### 2.2 回收方法区

Java虚拟机规范中确实说过可以不要求虚拟机在方法区中实现垃圾回收，而且在方法区中进行垃圾回收的“性价比”一般比较低，方法区的垃圾收集主要回收两部分内容：废弃的常量和无用的类。

废弃的常量，以常量池中字面量的回收为例，假如一个字符串“abc”已经进入常量池中，但是当前系统已经没有任何一个String对象叫做“abc”的，也没有任何其他地方引用这个字面量，这个“abc”常量就会被清理出常量池。

判断一个无用的类需要同时满足下面3个条件才能算是“无用的类”：

1. 该类的所有实例都已经被回收
2. 加载该类的ClassLoader已经被回收
3. 该类对应的java.lang.Class对象已经没有任何地方被引用，无法在任何地方通过反射访问该类的方法。