【须知】
①达性分析算法是需要一个理论上的前提的：该算法的全过程都需要基于一个能保障一致性的快照中才能够分析，这意味着必须全程冻结用户线程的运行。
②【标记】阶段是所有使用可达性分析算法的垃圾回收器都有的阶段。
③遍历GCRoot【根节点枚举】，在OopMap的加持下很快；沿着GCRoot遍历对象图【添加标记】，很慢；
④不采取措施的情况：用户线程与GC线程并行，便会产生【浮动垃圾】与【对象消失】，根本原因是【引用关系变化】

jvm怎么判断哪些对象应该回收呢?
引用计数算法【缺：无法解决循环引用】、可达性分析算法【缺：停止用户线程】。

怎么解决可达性算法的缺点?
并发标记：让【GC线程】和【用户线程】同时运行；

并发标记时会有什么问题?
【浮动垃圾 - 活而死 - 标活收死】：标记时活着，后因【引用关系变化】导致死了，GC时不回收； 【尚可接受】
【对象消失 - 死而活 - 标死收活】：标记时死了，后因【引用关系变化】导致活了，GC时被回收； 【不可接受】

怎么解决对象消失现象?
记录变化的引用关系，待扫描完成后，执行GC前，先执行记录内容；
增量更新： 记录insert，黑色对象一旦插入了指向白色对象的引用之后，它就变回了灰色对象。

原始快照： 记录delete，无论引用关系删除与否，都会按照刚刚开始扫描那一刻的对象图快照开进行搜索。

这个记录是怎么实现的?
【写屏障】可以看作虚拟机层面对【引用类型字段赋值】这个动作的AOP切面；

只是补充两点：
1.这里的写屏障和我们常说的为了解决并发乱序执行问题的"内存屏障"不是一码事，需要区分开来。
2.写屏障可以看作虚拟机层面对"引用类型字段赋值"这个动作的AOP切面，在引用对象赋值时会产生一个环形通知，供程序执行额外的动作，也就是说赋值的前后都在写屏障的覆盖范畴内。在赋值前的部分的写屏障叫做写前屏障(Pre-Write Barrier)，在赋值后的则叫作写后屏障(Post-Write Barrier)。
所以，经过简单的推导我们可以知道：
增量更新用的是写后屏障(Post-Write Barrier)，记录了所有新增的引用关系。
原始快照用的是写前屏障(Pre-Write Barrier)，将所有即将被删除的引用关系的旧引用记录下来。