1. 三色标记法简介

一开始所有对象都是白色对象，从程序根节点（全局变量和栈）出发，遍历可达对象（遍历一次，不是递归），可达对象染色为灰色。接着遍历灰色对象，将灰色对象的可达对象染为灰色，结束后将自身染为黑色，最后剩下的所有白色对象就是可回收对象。

2. 假如没有STW（stop the world）

如果gc标记过程中没有STW的保护，程序也同时在运行。如果此时出现：

- 一个黑色对象引用了一个白色对象
- 这个白色对象被引用它的灰色对象解除引用

一次gc过程中黑色对象是不会被再次遍历的，那么可以预想得到，这个白色对象将会被错误的回收。如果给gc标记过程全程加上STW，这会导致用户态代码长时间的业务停滞，这是不可接受的。但是上述的两个假设比如同时满足才会出现对象错误被回收的问题，如果能够破坏其中一个假设，使其不成立，就不会出现这个问题了。go团队因此提出了混合写屏障机制来减少STW的时间，达到优化gc算法的目的。

3. 写屏障机制

go团队提出了两种方式来限制gc标记过程中可能会出现的两种情况。

- 强三色不变式：强制性的不允许黑色对象引用白色对象。
- 弱三色不变式：黑色对象可以用引用白色对象，但是必须有一个灰色对象的可达链路能够访问到这个白色对象。

go团队用插入屏障和删除屏障来实现强弱三色不变式。

- 插入屏障：对象被引用时触发的机制。具体操作是在A对象引用B对象的时候，B对象被标记为灰色。这个方式满足强三色不变式，因为这个白色对象被引用时就直接染色为灰色，不会被回收。为了保证“栈”性能，插入屏障不在“栈”上使用。在清除白色对象前，用STW对“栈”空间进行重新遍历，保证了“栈”上的变量也满足强三色不变式。
- 删除屏障：对象被解除引用时触发的机制。具体操作是被删除的对象，如果自身为灰色或者白色，那么被标记为灰色（保护灰色对象到白色对象的路径不会断）。删除屏障的缺点是当前gc过程中被删除的对象必须等到下一轮才被清除，这是为了保护白色对象的链路不在当前gc过程中被切断，避免了有黑色对象引用白色对象时被错误回收。

4. 混合写屏障机制

go在1.8之后启用了混合写屏障机制保障gc算法。 具体操作如下：

1. GC开始就将“栈”上的对象全部扫描并标记为黑色（之后不再进行第二次重复扫描，无需STW）
2. GC期间，任何在栈上创建的新对象，均为黑色
3. 被删除的对象标记为灰色
4. 被添加的对象标记为灰色

这种混合写屏障机制满足变形的弱三色不变式（结合了插入、删除写屏障两者的优点）