为什么java虚拟机在进行GC操作两次标记时，要stop the world?

1、CMS及其执行过程?

CMS，全称Concurrent Mark and Sweep，用于对年老代进行回收，目标是尽量减少应用的暂停时间，减少full gc发生的机率，利用和应用程序线程并发的垃圾回收线程来标记清除年老代。CMS并非没有暂停，而是用两次短暂停来替代串行标记整理算法的长暂停。

内外的设置正常收集周期是这样的：

1）CMS-initial-mark 初始标记

2）CMS-concurrent-mark 并发标记的

3）CMS-concurrent-preclean 执行预清理　　注：相当于两次concurrent-mark. 因为上一次c mark，太长.会有很多 changed object 出现.先干掉这波.到最好的 stop the world 的 remark 阶段,changed object 会少很多.

4）CMS-concurrent-abortable-preclean 执行可中止预清理

5）CMS-remark 重新标记

6）CMS-concurrent-sweep 并发清除

7）CMS-concurrent-reset 并发重设状态等待下次CMS的触发

其中，CMS-initial-mark和CMS-remark会stop-the-world。

2、为什么 CMS两次标记时要 stop the world?

我们知道垃圾回收首先是要经过标记的。对象被标记后就会根据不同的区域采用不同的收集方法。看上去很完美的一件事情，其实并不然。

大家有没有想过一件事情，当虚拟机完成两次标记后，便确认了可以回收的对象。但是，垃圾回收并不会阻塞我们程序的线程，他是与当前程序并发执行

的。

所以问题就出在这里，当GC线程标记好了一个对象的时候，此时我们程序的线程又将该对象重新加入了“关系网”中，当执行二次标记的时候，该对象也没有重写finalize()方法，因此回收的时候就会回收这个不该回收的对象。

虚拟机的解决方法就是在一些特定指令位置设置一些“安全点”，当程序运行到这些“安全点”的时候就会暂停所有当前运行的线程（Stop The World 所以叫STW），暂停后再找到“GC Roots”进行关系的组建，进而执行标记和清除。

这些特定的指令位置主要在：

1、循环的末尾

2、方法临返回前 / 调用方法的call指令后

3、可能抛异常的位置