1.旧生代空间不足

旧生代空间只有在新生代对象转入及创建为大对象、大数组时才会出现不足的现象，当执行Full GC后空间仍然不足，则抛出如下错误：

java.lang.OutOfMemoryError:java heap space

为避免以上两种状况引起的Full GC，调优时应尽量做到让对象在Minor GC阶段被回收，让对象在新生代多存活一段时间及不要创建过大的对象及数组。

2.永久代空间满

永久代中存放的为一个类的信息，当系统中要加载的类、反射的类和调用的方法较多时，永久代可能会被占满，在未配置为采用CMS GC的情况下会执行Full GC。如果经过Full GC仍然回收不了，那么JVM会抛出如下错误信息：

java.lang.OutOfMemoryError:PermGen space

为避免永久代占满造成Full GC现象，可采用的方法为增大永久代空间或转为CMS GC。

3.CMS GC（并发收集器）时出现promotion failed和concurrent mode failure

对于采用CMS进行旧生代GC的程序而言，尤其要注意GC日志中是否有promotion failed和concurrent mode failure两种状况，当这两种状况出现时可能会触发Full GC。

promotion failed是在进行Minor GC时，survivor space放不下，对象只能放入旧生代，而此时旧生代也放不下造成的。concurrent mode failure是在执行CMS GC的过程中同时有对象要放入旧生代，而此时旧生代空间不足造成的。

应对措施：增大survivor space、旧生代空间或者调低触发并发GC的比率。

4.统计得到的Minor GC晋升到旧生代的平均大小大于旧生代的剩余空间

这是一个较为复杂的触发情况，Hotspot为了避免由于新生代对象晋升到旧生代导致旧生代空间不足的现象，在进行Minor GC时，做了一个判断，如果之前统计所得到的Minor GC晋升到旧生代的平均大小大于旧生代的剩余空间，那么就直接触发Full GC。

例如程序第一次触发Minor GC后，有6MB的对象晋升到旧生代，那么当下一次Minor GC发生时，首先检查旧生代的剩余空间是否大于6MB，如果大于6MB，则执行Full GC。

CMS收集器：

M:标记对象，GC必须记住哪些对象可达，以便删除不可达的对象

S:删除未可达的对象并释放他们的内存

CMS是一种以最短停顿时间为目标的收集器，使用CMS并不能达到GC效率最高，但它尽可能降低GC时服务的停顿时间。

CMS垃圾回收整个过程分为六个步骤

1.初始标记

为了手机应用程序的对象引用需要暂停应用线程，该阶段完成后，应用程序线程再次启动。

2.并发标记

从第一阶段收集的对象引用开始，遍历所有其他的对象引用。

3.并发预处理

改变当运行第二阶段，由应用程序产生的对象引用，以更新第二阶段的结果

4.重新标记

由于第三个阶段是并发的，对象引用可能会发生进一步改变，因此应用程序线程会再一次被暂停以跟新这些变化。

5.并发清理

所有不再被引用的对象将从堆里清除掉

6.并发重置

收集器做一些收尾的工作，以便下一次GC周期能有一个干净的状态。