jvm内存：

线程私有的{

pc程序计数器：

Java虚拟机栈：

本地方法栈：

}；

线程共享的{

堆：

方法区：方法区大小不是固定的，jvm可根据需要动态调整。方法区主要存放类信息、常量、静态变量、编译后的代码。

控制方法区大小：减少程序中class数量、尽量使用较少的静态变量。

方法区溢出：加载很多类会有可能出现，GC不会在主程序运行期对此区域进行清理，可通过设置jvm启动参数解决：

-XX:MaxPermSize=256m

}

垃圾回收算法：

标记-清楚算法：老年代

标记-整理算法：老年代常用

复制算法：新生代常用

Minor GC：从新生代回收内存，关键是Eden区内存不足，造成不足的原因是Java对象大部分是朝生夕死(java局部对象)，

而死掉的对象就需要在合适的时机被JVM回收

Major GC：从老年代回收内存，一般比Minor GC慢10倍以上。

Full GC：对整个堆来说的，出现Full GC通常伴随至少一次Minor GC，但非绝对。Full GC被触发的时候：老年代内存不足；

持久代内存不足；统计得到的Minor GC晋升到老年代平均大小大于老年代空间。

JVM调优的原因以及导致Full GC的原因

1.对JVM内存的系统级的调优主要的目的是减少GC的频率和Full GC的次数。

2.导致Full GC的原因

1)年老代（Tenured）被写满

调优时尽量让对象在新生代GC时被回收、让对象在新生代多存活一段时间和不要创建过大的对象及数组避免直接在旧生代创建对象 。

2)持久代Pemanet Generation空间不足

增大Perm Gen空间，避免太多静态对象 ， 控制好新生代和旧生代的比例

3)System.gc()被显示调用

垃圾回收不要手动触发，尽量依靠JVM自身的机制

3.系统崩溃前的一些现象：

每次垃圾回收的时间越来越长，由之前的10ms延长到50ms左右，FullGC的时间也有之前的0.5s延长到4、5s

FullGC的次数越来越多，最频繁时隔不到1分钟就进行一次FullGC

年老代的内存越来越大并且每次FullGC后年老代没有内存被释放

之后系统会无法响应新的请求，逐渐到达OutOfMemoryError的临界值，这个时候就需要分析JVM内存快照dump。

注：如果满足下面的指标，则一般不需要进行GC：

Minor GC执行时间不到50ms；

Minor GC执行不频繁，约10秒一次；

Full GC执行时间不到1s；

Full GC执行频率不算频繁，不低于10分钟1次；

JVM调优参数参考

1.针对JVM堆的设置，一般可以通过-Xms -Xmx限定其最小、最大值，为了防止垃圾收集器在最小、最大之间收缩堆而产生额外的时间，通常把最大、最小设置为相同的值;

2.年轻代和年老代将根据默认的比例（1：2）分配堆内存， 可以通过调整二者之间的比率NewRadio来调整二者之间的大小，也可以针对回收代。

比如年轻代，通过 -XX:newSize -XX:MaxNewSize来设置其绝对大小。同样，为了防止年轻代的堆收缩，我们通常会把-XX:newSize -XX:MaxNewSize设置为同样大小。

3.年轻代和年老代设置多大才算合理

1）更大的年轻代必然导致更小的年老代，大的年轻代会延长普通GC的周期，但会增加每次GC的时间；小的年老代会导致更频繁的Full GC

2）更小的年轻代必然导致更大年老代，小的年轻代会导致普通GC很频繁，但每次的GC时间会更短；大的年老代会减少Full GC的频率

如何选择应该依赖应用程序对象生命周期的分布情况： 如果应用存在大量的临时对象，应该选择更大的年轻代；如果存在相对较多的持久对象，年老代应该适当增大。

但很多应用都没有这样明显的特性。

在抉择时应该根 据以下两点：

（1）本着Full GC尽量少的原则，让年老代尽量缓存常用对象，JVM的默认比例1：2也是这个道理 。

（2）通过观察应用一段时间，看其他在峰值时年老代会占多少内存，在不影响Full GC的前提下，根据实际情况加大年轻代，比如可以把比例控制在1：1。

但应该给年老代至少预留1/3的增长空间。