# JVM优化

## JVM优化

先声明，我们这次指的JVM是HotSpot

JVM的优化主要原则：

1. 防止出现OutOfMemoryError异常。

出现OutOfMemoryError异常的原因有：

* 1. 虚拟机栈&本地方法栈（HotSpot把这二者合二为一）溢出。
  2. 永久代（也是方法区）溢出。
  3. Java堆溢出。
  4. 直接内存溢出。虽然直接内存与JVM没关系，但是依然受到物理内存的限制，一旦JVM各个内存区与直接内存总和超过硬件与操作系统的限制，动态扩展的时候还是会报异常。

1. 防止出现StackOverflowError异常。
2. 防止因为频繁GC而导致程序失去响应。

对应的解决方案：

1. OutOfMemoryError异常：
   1. 栈空间，设置参数-Xss。
   2. 方法区，设置参数-XX:PermSize与-XX:MaxPermSize。
   3. 堆，设置参数-Xmx与-Xms相等，避免每次GC之后JVM频繁分配内存。

设置新生代的参数，按优先级大小排列：-XX:NewSize与-XX:MaxNewSize>-Xmn>--XX:NewRatio(新生代与老年代的比值)。

设置老年代的参数：-XX:PermSize与-XX:MaxPermSize。

* 1. 直接内存：-XX:MaxDirectMemorySize。

1. StackOverflowError异常：

这个主要原因在编码上，慎用递归，避免死循环。

1. 频繁GC会导致程序失去响应，所以我们希望的目的就是：每次GC的时间越短越好，次数越少越好。

我们可以做以下布置（这只是建议，实际工作中要按照真实情况选择）：

3.1、少用大对象。这样就防止老年代空间被占满，防止总是Full GC。

3.2、大对象不可避免的情况下，让其直接进入老年代：-XX:PetenureSizeThreshold。

3.3、在不影响老年代空间的情况下，尽量把对象留在新生代。可以把新生代搞大一点，也可以把对象进入老年代的年龄搞大一点：-XX:MaxTenuringThreshold。

3.4、选择合适的垃圾收集器。有关垃圾收集器的介绍回头有空写一篇。

## 垃圾收集器

串行的就不说了。

**并行收集器：**由串行的单线程，改为多线程并行垃圾回收。适用于“对吞吐量有高要求”，对响应时间无要求的应用，后台处理，科学运算等。

**并发收集器：**在应用不停止的情况下进行垃圾收集。适用于“对响应时间要求较高”的应用，比如Web服务/后台服务等。