对于JVM的内存写过的文章已经有点多了，而且有点烂了，不过说那么多大多数在解决OOM的情况，于此，本文就只阐述这个内容，携带一些分析和理解和部分扩展内容，也就是JVM宕机中的一些问题，OK，下面说下OOM的常见情况：

a. 第一类内存溢出，也是大家认为最多，第一反应认为是的内存溢出，就是堆栈溢出：

那什么样的情况就是堆栈溢出呢？当你看到下面的关键字的时候它就是堆栈溢出了：

Java.lang.OutOfMemoryError: .....Java heap space.....

也就是当你看到heap相关的时候就肯定是堆栈溢出了，此时如果代码没有问题的情况下，适当调整-Xmx和-Xms是可以避免的，不过一定是代码没有问题的前提，为什么会溢出呢，要么代码有问题，要么访问量太多并且每个访问的时间太长或者数据太多，导致数据释放不掉，因为垃圾回收器是要找到那些是垃圾才能回收，这里它不会认为这些东西是垃圾，自然不会去回收了；主意这个溢出之前，可能系统会提前先报错关键字为：

java.lang.OutOfMemoryError:GC over head limit exceeded

这种情况是当系统处于高频的GC状态，而且回收的效果依然不佳的情况，就会开始报这个错误，这种情况一般是产生了很多不可以被释放的对象，有可能是引用使用不当导致，或申请大对象导致，但是java heap space的内存溢出有可能提前不会报这个错误，也就是可能内存就直接不够导致，而不是高频GC.

a. 第二类内存溢出，PermGen的溢出，或者PermGen 满了的提示，你会看到这样的关键字：

关键信息为：

java.lang.OutOfMemoryError: PermGen space

原因：系统的代码非常多或引用的第三方包非常多、或代码中使用了大量的常量、或通过intern注入常量、或者通过动态代码加载等方法，导致常量池的膨胀，虽然JDK 1.5以后可以通过设置对永久带进行回收，但是我们希望的是这个地方是不做GC的，它够用就行，所以一般情况下今年少做类似的操作，所以在面对这种情况常用的手段是：增加-XX:PermSize和-XX:MaxPermSize的大小。

a. 第三类内存溢出：在使用ByteBuffer中的allocateDirect()的时候会用到，很多javaNIO的框架中被封装为其他的方法

溢出关键字：

java.lang.OutOfMemoryError: Direct buffer memory

如果你在直接或间接使用了ByteBuffer中的allocateDirect方法的时候，而不做clear的时候就会出现类似的问题，常规的引用程序IO输出存在一个内核态与用户态的转换过程，也就是对应直接内存与非直接内存，如果常规的应用程序你要将一个文件的内容输出到客户端需要通过OS的直接内存转换拷贝到程序的非直接内存（也就是heap中），然后再输出到直接内存由操作系统发送出去，而直接内存就是由OS和应用程序共同管理的，而非直接内存可以直接由应用程序自己控制的内存，jvm垃圾回收不会回收掉直接内存这部分的内存，所以要注意了哦。

如果经常有类似的操作，可以考虑设置参数：-XX:MaxDirectMemorySize

a. 第四类内存溢出错误：

溢出关键字：

java.lang.StackOverflowError

这个参数直接说明一个内容，就是-Xss太小了，我们申请很多局部调用的栈针等内容是存放在用户当前所持有的线程中的，线程在jdk 1.4以前默认是256K，1.5以后是1M，如果报这个错，只能说明-Xss设置得太小，当然有些厂商的JVM不是这个参数，本文仅仅针对Hotspot VM而已；不过在有必要的情况下可以对系统做一些优化，使得-Xss的值是可用的。

a. 第五类内存溢出错误：

溢出关键字：

java.lang.OutOfMemoryError: unable to create new native thread

上面第四种溢出错误，已经说明了线程的内存空间，其实线程基本只占用heap以外的内存区域，也就是这个错误说明除了heap以外的区域，无法为线程分配一块内存区域了，这个要么是内存本身就不够，要么heap的空间设置得太大了，导致

了剩余的内存已经不多了，而由于线程本身要占用内存，所以就不够用了，说明了原因，如何去修改，不用我多说，你懂的。

a. 第六类内存溢出：

溢出关键字

java.lang.OutOfMemoryError: request {} byte for {}out of swap

这类错误一般是由于地址空间不够而导致。

六大类常见溢出已经说明JVM中99%的溢出情况，要逃出这些溢出情况非常困难，除非一些很怪异的故障问题会发生，比如由于物理内存的硬件问题，导致了code cache的错误（在由byte code转换为native code的过程中出现，但是概率极低），这种情况内存 会被直接crash掉，类似还有swap的频繁交互在部分系统中会导致系统直接被crash掉，OS地址空间不够的话，系统根本无法启动，呵呵；JNI的滥用也会导致一些本地内存无法释放的问题，所以尽量避开JNI；socket连接数据打开过多的socket也会报类似：IOException: Too many open files等错误信息。

JNI就不用多说了，尽量少用，除非你的代码太牛B了，我无话可说，呵呵，这种内存如果没有在被调用的语言内部将内存释放掉（如C语言），那么在进程结束前这些内存永远释放不掉，解决办法只有一个就是将进程kill掉。

另外GC本身是需要内存空间的，因为在运算和中间数据转换过程中都需要有内存，所以你要保证GC的时候有足够的内存哦，如果没有的话GC的过程将会非常的缓慢。

顺便这里就提及一些新的**CMS GC的内容和策略**（有点乱，每次写都很乱，但是能看多少看多少吧）：

首先我再写一次一前博客中的已经写过的内容，就是很多参数没啥建议值，建议值是自己在现场根据实际情况科学计算和测试得到的综合效果，建议值没有绝对好的，而且默认值很多也是有问题的，因为不同的版本和厂商都有很大的区别，

默认值没有永久都是一样的，就像-Xss参数的变化一样，要看到你当前的java程序heap的大致情况可以这样看看（以下参数是随便设置的，并不是什么默认值）：

```
$sudo jmap -heap `pgrep java`
Attaching to process ID 4280, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 19.1-b02
using thread-local object allocation.
Parallel GC with 8 thread(s)
Heap Configuration:
MinHeapFreeRatio = 40
MaxHeapFreeRatio = 70
MaxHeapSize = 1073741824 (1024.0MB)
NewSize = 134217728 (128.0MB)
MaxNewSize = 134217728 (128.0MB)
OldSize = 5439488 (5.1875MB)
NewRatio = 2
SurvivorRatio = 8
PermSize = 134217728 (128.0MB)
MaxPermSize = 268435456 (256.0MB)
Heap Usage:
PS Young Generation
Eden Space:
capacity = 85721088 (81.75MB)
used = 22481312 (21.439849853515625MB)
free = 63239776 (60.310150146484375MB)
26.22611602876529% used
From Space:
capacity = 24051712 (22.9375MB)
```

used = 478488 (0.45632171630859375MB)

free = 23573224 (22.481178283691406MB)

1.9894134770946867% used

To Space:

capacity = 24248320 (23.125MB)

used = 0 (0.0MB)

free = 24248320 (23.125MB)

0.0% used

PS Old Generation

capacity = 939524096 (896.0MB)

used = 16343864 (15.586723327636719MB)

free = 923180232 (880.4132766723633MB)

1.7395896571023124% used

PS Perm Generation

capacity = 134217728 (128.0MB)

used = 48021344 (45.796722412109375MB)

free = 86196384 (82.20327758789062MB)

35.77868938446045% used

注：sudo是需要拿到管理员权限，如果你的系统权限很大那么就不需要了，最后的grep java那个内容如果不对，可以直接通过jps或者ps命令将和java相关的进程号直接写进去，如：java -map 4280，这个参数其实完全可以通过jstat工具来替代，而且看到的效果更加好，这个参数在线上应用中，尽量少用（尤其是高并发的应用中），可能会触发JVM的bug，导致应用挂起；在jvm 1.6u14后可以编写任意一段程序，然后在运行程序的时候，增加参数为：-XX:+PrintFlagsFinal来输出当前JVM中运行时的参数值，或者通过jinfo来查看，jinfo是非常强大的工具，可以对部分参数进行动态修改，当然内存相关的东西是不能修改的，只能增加一些不是很相关的参数，有关JVM的工具使用，后续文章中如果有机会我们再来探讨，不是本文的重点；补充：关于参数的默认值对不同的JVM版本、不同的厂商、运行于不同的环境（一般和位数有关系）默认值会有区别。

OK，再说下反复的一句，没有必要的话就不要乱设置参数，参数不是拿来玩的，默认的参数对于这门JDK都是有好处的，关键是否适合你的应用场景，一般来讲你常规的只需要设置以下几个参数就可以了：

-server 表示为服务器端，会提供很多服务器端默认的配置，如并行回收，而服务器上一般这个参数都是默认的，所以都是可以省掉，与之对应的还有一个-client参数，一般在64位机器上，JVM是默认启动-server参数，也就是默认启动并行GC的，但是是ParallelGC而不是ParallelOldGC，两者算法不同（后面会简单说明下），而比较特殊的是windows 32位上默认是-client，这两个的区别不仅仅是默认的参数不一样，在jdk包下的jre包下一般会包含client和server包，下面分别对应启动的动态链接库，而真正看到的java、javac等相关命令指示一个启动导向，它只是根据命令找到对应的JVM并传入jvm中进行启动，也就是看到的java.exe这些文件并不是jvm；说了这么多，最终总结一下就是，-server和-client就是完全不同的两套VM，一个用于桌面应用，一个用于服务器的。

-Xmx 为Heap区域的最大值

-Xms 为Heap区域的初始值，线上环境需要与-Xmx设置为一致，否则capacity的值会来回飘动，飘得你心旷神怡，你懂的。

-Xss（或-ss） 这个其实也是可以默认的，如果你真的觉得有设置的必要，你就改下吧，1.5以后是1M的默认大小（指一个线程的native空间），如果代码不多，可以设置小点来让系统可以接受更大的内存。注意，还有一个参数是-XX:ThreadStackSize，这两个参数在设置的过程中如果都设置是有冲突的，一般按照JVM常理来说，谁设置在后面，就以谁为主，但是最后发现如果是在1.6以上的版本，-Xss设置在后面的确都是以-Xss为主，但是要是-XX:ThreadStackSize设置在后面，主线程还是为-Xss为主，而其它线程以-XX:ThreadStackSize为主，主线程做了一个特殊判定处理；单独设置都是以本身为主，-Xss不设置也不会采用其默认值，除非两个都不设置会采用-Xss的默认值。另外这个参数针对于hotspot的vm，在IBM的jvm中，还有一个参数为-Xoss，主要原因是IBM在对栈的处理上有操作数栈和方法栈等各种不同的栈种类，而hotspot不管是什么栈都放在一个私有的线程内部的，不区分是什么栈，所以只需要设置一个参数，而IBM的J9不是这样的；有关栈上的细节，后续我们有机会专门写文章来说明。

-XX:PermSize与-XX:MaxPermSize两个包含了class的装载的位置，或者说是方法区（但不是本地方法区），在Hotspot默认情况下为64M，主意全世界的JVM只有hostpot的VM才有Perm的区域，或者说只有hotspot才有对用户可以设置的这块区域，其他的JVM都没有，其实并不是没有这块区域，而是这块区域没有让用户来设置，其实这块区域本身也不应该让用户来设置，我们也没有一个明确的说法这块空间必须要设置多大，都是拍脑袋设置一个数字，如果发布到线上看下如果用得比较多，就再多点，如果用的少，就减少点，而这块区域和性能关键没有多大关系，只要能装下就OK，并且时不时会因为Perm不够而导致Full GC，所以交给开发者来调节这个参数不知道是怎么想的；所以Oracle将在新一代JVM中将这个区域彻底删掉，也就是对用户透明，G1的如果真正稳定起来，以后JVM的启动参数将会非常简单，而且理论上管理再大的内存也是没有问题的，其实G1（garbage first，一种基于region的垃圾收集回收器）已经在hotspot中开始有所试用，不过目前效果不好，还不如CMS呢，所以只是试用，G1已经作为oracle对JVM研发的最高重点，CMS自现在最高版本后也不再有新功能（可以修改bug），该项目已经进行5年，尚未发布正式版，CMS是四五年前发布的正式版，但是是最近一两年才开始稳定，而G1的复杂性将会远远超越CMS，所以要真正使用上G1还有待考察，全世界目前只有IBM J9真正实现了G1论文中提到的思想（论文于05年左右发表），IBM已经将J9应用于websphere中，但是并不代表这是全世界最好的jvm，全世界最好的jvm是Azul(无停顿垃圾回收算法和一个零开销的诊断/监控工具)，几乎可以说这个jvm是没有暂停的，在全世界很多顶尖级的公司使用，不过价格非常贵，不能直接使用，目前这个jvm的主导者在研究JRockit，而目前hotspot和JRockit都是Oracle的，所以他们可能会合并，所以我们应该对JVM的性能充满信心。

也就是说你常用的情况下只需要设置4个参数就OK了，除非你的应用有些特殊，否则不要乱改，那么来看看一些其他情况的参数吧：

先来看个不大常用的，就是大家都知道JVM新的对象应该说几乎百分百的在Eden里面，除非Eden真的装不下，我们不考虑这种变态的问题，因为线上环境Eden区

域都是不小的，来降低GC的次数以及全局 GC的概率；而JVM习惯将内存按照较为连续的位置进行分配，这样使得有足够的内存可以被分配，减少碎片，那么对于内存最后一个位置必然就有大量的征用问题，JVM在高一点的版本里面提出了为每个线程分配一些私有的区域来做来解决这个问题，而1.5后的版本还可以动态管理这些区域，那么如何自己设置和查看这些区域呢，看下英文全称为：Thread Local Allocation Buffer，简称就是：TLAB，即内存本地的持有的buffer，设置参数有：

-XX:+UseTLAB 启用这种机制的意思

-XX:TLABSize= 设置大小，也就是本地线程中的私有区域大小（只有这个区域放不下才会到Eden中去申请）。

-XX:+ResizeTLAB 是否启动动态修改

这几个参数在多CPU下非常有用。

-XX:+PrintTLAB 可以输出TLAB的内容。

下面再闲扯些其它的参数：

如果你需要对Yong区域进行并行回收应该如何修改呢？在jdk1.5以后可以使用参数：

-XX:+UseParNewGC

注意： 与它冲突的参数是：-XX:+UseParallelOldGC和-XX:+UseSerialGC，如果需要用这个参数，又想让整个区域是并行回收的，那么就使用-XX:+UseConcMarkSweepGC参数来配合，其实这个参数在使用了CMS后，默认就会启动该参数，也就是这个参数在CMS GC下是无需设置的，后面会提及到这些参数。

默认服务器上的对Full并行GC策略为（这个时候Yong空间回收的时候启动PSYong算法，也是并行回收的）：

-XX:+UseParallelGC

另外，在jdk1.5后出现一个新的参数如下，这个对Yong的回收算法和上面一样，对Old区域会有所区别，上面对Old回收的过程中会做一个全局的Compact，也就是全局的压缩操作，而下面的算法是局部压缩，为什么要局部压缩呢？是因为JVM发现每次压缩后再逻辑上数据都在Old区域的左边位置，申请的时候从左向右申请，那么生命力越长的对象就一般是靠左的，所以它认为左边的对象就是生命力很强，而且较为密集的，所以它针对这种情况进行部分密集，但是这两种算法mark阶段都是会暂停的，而且存活的对象越多活着的越多；而ParallelOldGC会进行部分压缩算法（主意一点，最原始的copy算法是不需要经过mark阶段，因为只需要找到一个或活着的就只需要做拷贝就可以，而Yong区域借用了Copy算法，只是唯一的区别就是传统的copy算法是采用两个相同大小的内存来拷贝，浪费空间为50%，所以分代的目标就是想要实现很多优势所在，认为新生代85%以上的对象都应该是死掉的，所以S0和S1一般并不是很大），该算法为jdk 1.5以后对于绝大部分应用的最佳选择。

-XX:+UseParallelOldGC

-XX:ParallelGCThread=12：并行回收的线程数，最好根据实际情况而定，因为线程多往往存在征用调度和上下文切换的开销；而且也并非CPU越多线程数也可以设置越大，一般设置为12就再增加用处也不大，主要是算法本身内部的征用会导致其线程的极限就是这样。

设置Yong区域大小：

-Xmn Yong区域的初始值和最大值一样大

-XX:NewSize和-XX:MaxNewSize如果设置以为一样大就是和-Xmn，在JRockit中会动态变化这些参数，根据实际情况有可能会变化出两个Yong区域，或者没有Yong区域，有些时候会生出来一个半长命对象区域；这里除了这几个参数外，还有一个参数是NewRatio是设置*Old/Yong*的倍数的，这几个参数都是有冲突的，服务器端建议是设置-Xmn就可以了，如果几个参数全部都有设置，-Xmn和-XX:NewSize与-XX:MaxNewSize将是谁设置在后面，以谁的为准，而-XX:NewSize -XX:MaxNewSize与-XX:NewRatio时，那么参数设置的结果可能会以下这样的（jdk 1.4.1后）：

min(MaxNewSize,max(NewSize, heap/(NewRatio+1))
-XX:NewRatio为Old区域为Yong的多少倍，间接设置Yong的大小，1.6中如果使用此参数，则默认会在适当时候被动态调整，具体请看下面参数UseAdaptiveSizepollcy 的说明。
三个参数不要同时设置，因为都是设置Yong的大小的。

-XX:SurvivorRatio：该参数为Eden与两个求助空间之一的比例，注意Yong的大小等价于Eden + S0 + S1，S0和S1的大小是等价的，这个参数为Eden与其中一个S区域的大小比例，如参数为8，那么Eden就占用Yong的80%，而S0和S1分别占用10%。
以前的老版本有一个参数为：-XX:InitialSurivivorRatio，如果不做任何设置，就会以这个参数为准，这个参数的默认值就是8，不过这个参数并不是Eden/Survivor的大小，而是Yong/Survivor，所以所以默认值8，代表每一个S区域的空间大小为Yong区域的12.5%而不是10%。另外顺便提及一下，每次大家看到GC日志的时候，GC日志中的每个区域的最大值，其中Yong的空间最大值，始终比设置的Yong空间的大小要小一点，大概是小12.5%左右，那是因为每次可用空间为Eden加上一个Survivor区域的大小，而不是整个Yong的大小，因为可用空间每次最多是这样大，两个Survivor区域始终有一块是空的，所以不会加上两个来计算。

-XX:MaxTenuringThreshold=15：在正常情况下，新申请的对象在Yong区域发生多少次GC后就会被移动到Old（非正常就是S0或S1放不下或者不太可能出现的Eden都放不下的对象），这个参数一般不会超过16（因为计数器从0开始计数，所以设置为15的时候相当于生命周期为16）。
要查看现在的这个值的具体情况，可以使用参数：-XX:+PrintTenuringDistribution

通过上面的jmap应该可以看出我的机器上的MinHeapFreeRatio和MaxHeapFreeRatio分别为40个70，也就是大家经常说的在GC后剩余空间小于40%

时capacity开始增大，而大于70%时减小，由于我们不希望让它移动，所以这两个参数几乎没有意义，如果你需要设置就设置参数为：

-XX:MinHeapFreeRatio=40

-XX:MaxHeapFreeRatio=70

JDK 1.6后有一个动态调节板块的，当然如果你的每一个板块都是设置固定值，这个参数也没有用，不过如果是非固定的，建议还是不要动态调整，默认是开启的，建议将其关掉，参数为：

-XX:+UseAdaptiveSizepollcy 建议使用-XX:-UseAdaptiveSizepollcy关掉，为什么当你的参数设置了NewRatio、Survivor、MaxTenuringThreshold这几个参数如果在启动了动态更新情况下，是无效的，当然如果你设置-Xmn是有效的，但是如果设置的比例的话，初始化可能会按照你的参数去运行，不过运行过程中会通过一定的算法动态修改，监控中你可能会发现这些参数会发生改变，甚至于S0和S1的大小不一样。

如果启动了这个参数，又想要跟踪变化，那么就使用参数：-
XX:+PrintAdaptiveSizePolicy

上面已经提到，javaNIO中通过Direct内存来提高性能，这个区域的大小默认是64M，在适当的场景可以设置大一些。

-XX:MaxDirectMemorySize

一个不太常用的参数：

-XX:+ScavengeBeforeFullGC 默认是开启状态，在full GC前先进行minor GC。

对于java堆中如果要设置大页内存，可以通过设置参数：

付：此参数必须在操作系统的内核支持的基础上，需要在OS级别做操作为：

echo 1024 > /proc/sys/vm/nr_hugepages

echo 2147483647 > /proc/sys/kernel/shmmax

-XX:+UseLargePages

-XX:LargePageSizeInBytes

此时整个JVM都将在这块内存中，否则全部不在这块内存中。

javaIO的临时目录设置
-Djava.io.tmpdir
jstack会去寻找/tmp/hsperfdata_admin下去寻找与进程号相同的文件，32位机器上是没有问题的，64为机器的是有BUG的，在jdk 1.6u23版本中已经修复了这个bug，如果你遇到这个问题，就需要升级JDK了。

还记得上次说的平均晋升大小吗，在并行GC时，如果平均晋升大小大于old剩余空间，则发生full GC，那么当小于剩余空间时，也就是平均晋升小于剩余空间，但是剩余空间小于eden + 一个survivor的空间时，此时就依赖于参数：
-XX:-HandlePromotionFailure
启动该参数时，上述情况成立就发生minor gc（YGC），大于则发生full gc（major gc）。

一般默认直接分配的对象如果大于Eden的一半就会直接晋升到old区域，但是也可以通过参数来指定：
-XX:PretenureSizeThreshold=2m 我个人不建议使用这个参数
也就是当申请对象大于这个值就会晋升到old区域。

传说中GC时间的限制，一个是通过比例限制，一个是通过最大暂停时间限制，但是GC时间能限制么，呵呵，在增量中貌似可以限制，不过不能限制住GC总体的时间，所以这个参数也不是那么关键。
-XX:GCTimeRatio=
-XX:MaxGCPauseMillis
-XX:GCTimeLimit
要看到真正暂停的时间就一个是看GCDetail的日志，另一个是设置参数看：
-XX:+PrintGCApplicationStoppedTime

有些人，有些人就是喜欢在代码里面里头写System.gc()，耍酷，这个不是测试程序是线上业务，这样将会导致N多的问题，不多说了，你应该懂的，不懂的话看下书吧，而RMI是很不听话的一个鸟玩意，EJB的框架也是基于RMI写的，RMI为什么不听话呢，就是它自己在里面非要搞个System.gc()，哎，为了放置频繁的做，频繁的做，你就将这个命令的执行禁用掉吧，当然程序不用改，不然那些EJB都跑步起来了，呵呵：

-XX:+DisableExplicitGC 默认是没有禁用掉，写成+就是禁用掉的了，但是有些时候在使用allocateDirect的时候，很多时候还真需要System.gc来强制回收这块资源。

内存溢出时导出溢出的错误信息：

-XX:+HeapDumpOnOutOfMemoryError

-XX:HeapDumpPath=/home/xieyu/logs/ 这个参数指定导出时的路径，不然导出的路径就是虚拟机的目标位置，不好找了，默认的文件名是：java_pid<进程号>.hprof，这个文件可以类似使用jmap -dump:file=....,format=b 来dump类似的内容，文件后缀都是hprof，然后下载mat工具进行分析即可（不过内存有多大dump文件就多大，而本地分析的时候内存也需要那么大，所以很多时候下载到本地都无法启动是很正常的），后续文章有机会我们来说明这些工具，另外jmap -dump参数也不要经常用，会导致应用挂起哦;另外此参数只会在第一次输出OOM的时候才会进行堆的dump操作（java heap的溢出是可以继续运行再运行的程序的，至于web应用是否服务要看应用服务器自身如何处理，而c heap区域的溢出就根本没有dump的机会，因为直接就宕机了，目前系统无法看到c heap的大小以及内部变化，要看大小只能间接通过看JVM进程的内存大小（top或类似参数），这个大小一般会大于heap+perm的大小，多余的部分基本就可以认为是c heap的大小了，而看内部变化呢只有google perftools可以达到这个目的），如果内存过大这个dump操作将会非常长，所以hotspot如果以后想管理大内存，这块必须有新的办法出来。

最后，用dump出来的文件，通过mat分析出来的结果往往有些时候难以直接确定到底哪里有问题，可以看到的维度大概有：那个类使用的内存最多，以及每一个线程使用的内存，以及线程内部每一个调用的类和方法所使用的内存，但是很多

时候无法判定到底是程序什么地方调用了这个类或者方法，因为这里只能看到最终消耗内存的类，但是不知道谁使用了它，一个办法是扫描代码，但是太笨重，而且如果是jar包中调用了就不好弄了，另一种方法是写agent，那么就需要相应的配合了，但是有一个非常好的工具就是btrace工具（jdk 1.7貌似还不支持），可以跟踪到某个类的某个方法被那些类中的方法调用过，那这个问题就好说了，只要知道开销内存的是哪一个类，就能知道谁调用过它，OK，关于btrace的不是本文重点，网上都有，后续文章有机会再探讨，

原理：

No performance impact during runtime（无性能影响）

Dumping a -Xmx512m heap

Create a 512MB .hprof file（512M内存就dump出512M的空间大小）

JVM is “dead” during dumping（死掉时dump）

Restarting JVM during this dump will cause unusable .hprof file（重启导致文件不可用）

注明的NUMA架构，在JVM中开始支持，当然也需要CPU和OS的支持才可以，需要设置参数为：

-XX:+UseNUMA 必须在并行GC的基础上才有的

老年代无法分配区域的最大等待时间为(默认值为0，但是也不要去动它)：

-XX:GCExpandToAllocateDelayMillis

让JVM中所有的set和get方法转换为本地代码：

-XX:+UseFastAccessorMethods

以时间戳输出Heap的利用率

-XX:+PrintHeapUsageOverTime

在64bit的OS上面（其实一般达不到57位左右），由于指针会放大为8个byte，所以会导致空间使用增加，当然，如果内存够大，就没有问题，但是如果升级到64bit系统后，只是想让内存达到4G或者8G，那么就完全可以通过很多指针压缩为4byte就OK了，所以在提供以下参数(本参数于jdk 1.6u23后使用，并自动开启，所以也不需要你设置，知道就OK)：

-XX:+UseCompressedOops 请注意：这个参数默认在64bit的环境下默认启动，但是如果JVM的内存达到32G后，这个参数就会默认为不启动，因为32G内存后，压缩就没有多大必要了，要管理那么大的内存指针也需要很大的宽度了。

后台JIT编译优化启动

-XX:+BackgroundCompilation

如果你要输出GC的日志以及时间戳，相关的参数有：

-XX:+PrintGCDetails 输出GC的日志详情，包含了时间戳

-XX:+PrintGCTimeStamps 输出GC的时间戳信息，按照启动JVM后相对时间的每次GC的相对秒值（毫秒在小数点后面），也就是每次GC相对启动JVM启动了多少秒后发生了这次GC

-XX:+PrintGCDateStamps输出GC的时间信息，会按照系统格式的日期输出每次GC的时间

-XX:+PrintGCTaskTimeStamps输出任务的时间戳信息，这个细节上比较复杂，后续有文章来探讨。

-XX:-TraceClassLoading 跟踪类的装载

-XX:-TraceClassUnloading 跟踪类的卸载

-XX:+PrintHeapAtGC 输出GC后各个堆板块的大小。

将常量信息GC信息输出到日志文件：

-Xloggc:/home/xieyu/logs/gc.log

现在面对大内存比较流行是是CMS GC（最少1.5才支持），首先明白CMS的全称是什么，不是传统意义上的内容管理系统（Content Management System）哈，第一次我也没看懂，它的全称是：Concurrent Mark Sweep，三个单词分别代表并发、标记、清扫（主意这里没有compact操作，其实CMS GC的确没有compact操作），也就是在程序运行的同时进行标记和清扫工作，至于它的原理前面有提及过，只是有不同的厂商在上面做了一些特殊的优化，比如一些厂商在标记根节点的过程中，标记完当前的根，那么这个根下面的内容就不会被暂停恢复运行了，而移动过程中，通过读屏障来看这个内存是不是发生移动，如果在移动稍微停一

下，移动过去后再使用，hotspot还没这么厉害，暂停时间还是挺长的，只是相对其他的GC策略在面对大内存来讲是不错的选择。

下面看一些CMS的策略（并发GC总时间会比常规的并行GC长，因为它是在运行时去做GC，很多资源征用都会影响其GC的效率，而总体的暂停时间会短暂很多很多，其并行线程数默认为：（上面设置的并行线程数 + 3）/ 4

付：CMS是目前Hotspot管理大内存最好的JVM，如果是常规的JVM，最佳选择为ParallelOldGC，如果必须要以响应时间为准，则选择CMS，不过CMS有两个隐藏的隐患：

1、CMS GC虽然是并发且并行运行的GC，但是初始化的时候如果采用默认值92%（JVM 1.5的白皮书上描述为68%其实是错误的，1.6是正确的），就很容易出现问题，因为CMS GC仅仅针对Old区域，Yong区域使用ParNew算法，也就是Old的CMS回收和Yong的回收可以同时进行，也就是回收过程中Yong有可能会晋升对象Old，并且业务也可以同时运行，所以92%基本开始启动CMS GC很有可能old的内存就不够用了，当内存不够用的时候，就启动Full GC，并且这个Full GC是串行的，所以如果弄的不好，CMS会比并行GC更加慢，为什么要启用串行是因为CMS GC、并行GC、串行GC的继承关系决定的，简单说就是它没办法去调用并行GC的代码，细节说后续有文章来细节说明），建议这个值设置为70%左右吧，不过具体时间还是自己决定。

2、CMS GC另一个大的隐患，其实不看也差不多应该清楚，看名字就知道，就是不会做Compact操作，它最恶心的地方也在这里，所以上面才说一般的应用都不使用它，它只有内存垃圾非常多，多得无法分配晋升的空间的时候才会出现一次compact，但是这个是Full GC，也就是上面的串行，很恐怖的，所以内存不是很大的，不要考虑使用它，而且它的算法十分复杂。

还有一些小的隐患是：和应用一起征用CPU（不过这个不是大问题，增加CPU即可）、整个运行过程中时间比并行GC长（这个也不是大问题，因为我们更加关心暂停时间而不是运行时间，因为暂停会影响非常多的业务）。

启动CMS为全局GC方法(注意这个参数也不能上面的并行GC进行混淆，Yong默认是并行的，上面已经说过

-XX:+UseConcMarkSweepGC

在并发GC下启动增量模式，只能在CMS GC下这个参数才有效。

-XX:+CMSIncrementalMode

启动自动调节duty cycle，即在CMS GC中发生的时间比率设置，也就是说这段时间内最大允许发生多长时间的GC工作是可以调整的。

-XX:+CMSIncrementalPacing

在上面这个参数设定后可以分别设置以下两个参数（参数设置的比率，范围为0-100）：

-XX:CMSIncrementalDutyCycleMin=0

-XX:CMSIncrementalDutyCycle=10

增量GC上还有一个保护因子（CMSIncrementalSafetyFactor），不太常用；CMSIncrementalOffset提供增量GC连续时间比率的设置；CMSExpAvgFactor为增量并发的GC增加权重计算。

-XX:CMSIncrementalSafetyFactor=

-XX:CMSIncrementalOffset=

-XX:CMSExpAvgFactor=

是否启动并行CMS GC（默认也是开启的）

-XX:+CMSParallelRemarkEnabled

要单独对CMS GC设置并行线程数就设置（默认也不需要设置）：

-XX:ParallelCMSThreads

对PernGen进行垃圾回收：

JDK 1.5在CMS GC基础上需要设置参数（也就是前提是CMS GC才有）：

-XX:+CMSClassUnloadingEnabled -XX:+CMSPermGenSweepingEnabled

1.6以后的版本无需设置：-XX:+CMSPermGenSweepingEnabled，注意，其实一直以来Full GC都会触发对Perm的回收过程，CMS GC需要有一些特殊照顾，虽然VM会对这块区域回收，但是Perm回收的条件几乎不太可能实现，首先需要这个类的

classloader必须死掉，才可以将该classloader下所有的class干掉，也就是要么全部死掉，要么全部活着；另外，这个classloader下的class没有任何object在使用，这个也太苛刻了吧，因为常规的对象申请都是通过系统默认的，应用服务器也有自己默认的classloader，要让它死掉可能性不大，如果这都死掉了，系统也应该快挂了。

CMS GC因为是在程序运行时进行GC，不会暂停，所以不能等到不够用的时候才去开启GC，官方说法是他们的默认值是68%，但是可惜的是文档写错了，经过很多测试和源码验证这个参数应该是在92%的时候被启动，虽然还有8%的空间，但是还是很可怜了，当CMS发现内存实在不够的时候又回到常规的并行GC，所以很多人在没有设置这个参数的时候发现CMS GC并没有神马优势嘛，和并行GC一个鸟样子甚至于更加慢，所以这个时候需要设置参数（这个参数在上面已经说过，启动CMS一定要设置这个参数）：
-XX:CMSInitiatingOccupancyFraction=70
这样保证Old的内存在使用到70%的时候，就开始启动CMS了；如果你真的想看看默认值，那么就使用参数：-XX:+PrintCMSInitiationStatistics 这个变量只有JDK 1.6可以使用 1.5不可以，查看实际值-XX:+PrintCMSStatistics；另外，还可以设置参数-XX:CMSInitiatingPermOccupancyFraction来设置Perm空间达到多少时启动CMS GC，不过意义不大。
JDK 1.6以后有些时候启动CMS GC是根据计算代价进行启动，也就是不一定按照你指定的参数来设置的，如果你不想让它按照所谓的成本来计算GC的话，那么你就使用一个参数：-XX:+UseCMSInitiatingOccupancyOnly，默认是false，它就只会按照你设置的比率来启动CMS GC了。如果你的程序中有System.gc以及设置了ExplicitGCInvokesConcurrent在jdk 1.6中，这种情况使用NIO是有可能产生问题的。

启动CMS GC的compation操作，也就是发生多少次后做一次全局的compaction：
-XX:+UseCMSCompactAtFullCollection
-XX:CMSFullGCsBeforeCompaction：发生多少次CMS Full GC，这个参数最好不要设置，因为要做compaction的话，也就是真正的Full GC是串行的，非常慢，

让它自己去决定什么时候需要做compaction。

-XX:CMSMaxAbortablePrecleanTime=5000 设置preclean步骤的超时时间，单位为毫秒，preclean为cms gc其中一个步骤，关于cms gc步骤比较多，本文就不细节探讨了。

并行GC在mark阶段，可能会同时发生minor GC，old区域也可能发生改变，于是并发GC会对发生了改变的内容进行remark操作，这个触发的条件是：
-XX:CMSScheduleRemarkEdenSizeThreshold
-XX:CMSScheduleRemarkEdenPenetration
即Eden区域多大的时候开始触发，和eden使用量超过百分比多少的时候触发，前者默认是2M，后者默认是50%。
但是如果长期不做remark导致old做不了，可以设置超时，这个超时默认是5秒，可以通过参数：
-XX:CMSMaxAbortablePrecleanTime
-XX:+ExplicitGCInvokesConcurrent 在显示发生GC的时候，允许进行并行GC。
-XX:+ExplicitGCInvokesConcurrentAndUnloadsClasses 几乎和上面一样，只不过多一个对Perm区域的回收而已。

补充：
其实JVM还有很多的版本，很多的厂商，与其优化的原则，随便举两个例子
hotspot在GC中做的一些优化（这里不说代码的编译时优化或运行时优化）：
Eden申请的空间对象由Old区域的某个对象的一个属性指向（也就是Old区域的这个空间不回收，Eden这块就没有必要考虑回收），所以Hotspot在CPU写上面，做了一个屏障，当发生赋值语句的时候（对内存来讲赋值就是一种写操作），如果发现是一个新的对象由Old指向Eden，那么就会将这个对象记录在一个卡片机里面，这个卡片机是有很多512字节的卡片组成，当在YGC过程中，就基本不会去移动或者管理这块对象（付：这种卡片机会在CMS GC的算法中使用，不过和这个卡片不是放在同一个地方的，也是CMS GC的关键，对于CMS GC的算法细节描述，后续文章我们单独说明）。

Old区域对于一些比较大的对象，JVM就不会去管理个对象，也就是compact过程中不会去移动这块对象的区域等等吧。

以上大部分参数为hotspot的自带关于性能的参数，参考版本为JDK 1.5和1.6的版本，很多为个人经验说明，不足以说明所有问题，如果有问题，欢迎探讨；另外，JDK的参数是不是就只有这些呢，肯定并不是，我知道的也不止这些，但是有些觉得没必要说出来的参数和一些数学运算的参数我就不想给出来了，比如像禁用掉GC的参数有神马意义，我们的服务器要是把这个禁用掉干个屁啊，呵呵，做测试还可以用这玩玩，让它不做GC直接溢出；还有一些什么计算因子啥的，还有很多复杂的数学运算规则，要是把这个配置明白了，就太那个了，而且一般情况下也没那个必要，JDK到现在的配置参数多达上500个以上，要知道完的话慢慢看吧，不过意义不大，而且要知道默认值最靠谱的是看源码而不是看文档，官方文档也只能保证绝大部是正确的，不能保证所有的是正确的。

本文最后追加在jdk 1.6u 24后通过上面说明的-XX:+PrintFlagsFinal输出的参数以及默认值（还是那句话，在不同的平台上是不一样的），输出的参数如下，可以看看JVM的参数是相当的多，参数如此之多，你只需要掌握关键即可，参数还有很多有冲突的，不要纠结于每一个参数的细节：

```
$java -XX:+PrintFlagsFinal
uintx AdaptivePermSizeWeight = 20 {product}
uintx AdaptiveSizeDecrementScaleFactor = 4 {product}
uintx AdaptiveSizeMajorGCDecayTimeScale = 10 {product}
uintx AdaptiveSizePausePolicy = 0 {product}
uintx AdaptiveSizePolicyCollectionCostMargin = 50 {product}
uintx AdaptiveSizePolicyInitializingSteps = 20 {product}
uintx AdaptiveSizePolicyOutputInterval = 0 {product}
uintx AdaptiveSizePolicyWeight = 10 {product}
uintx AdaptiveSizeThroughPutPolicy = 0 {product}
uintx AdaptiveTimeWeight = 25 {product}
```

```
bool AdjustConcurrency = false {product}
bool AggressiveOpts = false {product}
intx AliasLevel = 3 {product}
intx AllocatePrefetchDistance = -1 {product}
intx AllocatePrefetchInstr = 0 {product}
intx AllocatePrefetchLines = 1 {product}
intx AllocatePrefetchStepSize = 16 {product}
intx AllocatePrefetchStyle = 1 {product}
bool AllowJNIEnvProxy = false {product}
bool AllowParallelDefineClass = false {product}
bool AllowUserSignalHandlers = false {product}
bool AlwaysActAsServerClassMachine = false {product}
bool AlwaysCompileLoopMethods = false {product}
intx AlwaysInflate = 0 {product}
bool AlwaysLockClassLoader = false {product}
bool AlwaysPreTouch = false {product}
bool AlwaysRestoreFPU = false {product}
bool AlwaysTenure = false {product}
bool AnonymousClasses = false {product}
bool AssertOnSuspendWaitFailure = false {product}
intx Atomics = 0 {product}
uintx AutoGCSelectPauseMillis = 5000 {product}
intx BCEATraceLevel = 0 {product}
intx BackEdgeThreshold = 100000 {pd product}
bool BackgroundCompilation = true {pd product}
uintx BaseFootPrintEstimate = 268435456 {product}
intx BiasedLockingBulkRebiasThreshold = 20 {product}
intx BiasedLockingBulkRevokeThreshold = 40 {product}
intx BiasedLockingDecayTime = 25000 {product}
intx BiasedLockingStartupDelay = 4000 {product}
```

```
bool BindGCTaskThreadsToCPUs = false {product}
bool BlockOffsetArrayUseUnallocatedBlock = false {product}
bool BytecodeVerificationLocal = false {product}
bool BytecodeVerificationRemote = true {product}
intx CICompilerCount = 1 {product}
bool CICompilerCountPerCPU = false {product}
bool CITime = false {product}
bool CMSAbortSemantics = false {product}
uintx CMSAbortablePrecleanMinWorkPerIteration = 100 {product}
intx CMSAbortablePrecleanWaitMillis = 100 {product}
uintx CMSBitMapYieldQuantum = 10485760 {product}
uintx CMSBootstrapOccupancy = 50 {product}
bool CMSClassUnloadingEnabled = false {product}
uintx CMSClassUnloadingMaxInterval = 0 {product}
bool CMSCleanOnEnter = true {product}
bool CMSCompactWhenClearAllSoftRefs = true {product}
uintx CMSConcMarkMultiple = 32 {product}
bool CMSConcurrentMTEnabled = true {product}
uintx CMSCoordinatorYieldSleepCount = 10 {product}
bool CMSDumpAtPromotionFailure = false {product}
uintx CMSExpAvgFactor = 50 {product}
bool CMSExtrapolateSweep = false {product}
uintx CMSFullGCsBeforeCompaction = 0 {product}
uintx CMSIncrementalDutyCycle = 10 {product}
uintx CMSIncrementalDutyCycleMin = 0 {product}
bool CMSIncrementalMode = false {product}
uintx CMSIncrementalOffset = 0 {product}
bool CMSIncrementalPacing = true {product}
uintx CMSIncrementalSafetyFactor = 10 {product}
uintx CMSIndexedFreeListReplenish = 4 {product}
```

```
intx CMSInitiatingOccupancyFraction = -1 {product}
intx CMSInitiatingPermOccupancyFraction = -1 {product}
intx CMSIsTooFullPercentage = 98 {product}
double CMSLargeCoalSurplusPercent = {product}
double CMSLargeSplitSurplusPercent = {product}
bool CMSLoopWarn = false {product}
uintx CMSMaxAbortablePrecleanLoops = 0 {product}
intx CMSMaxAbortablePrecleanTime = 5000 {product}
uintx CMSOldPLABMax = 1024 {product}
uintx CMSOldPLABMin = 16 {product}
uintx CMSOldPLABNumRefills = 4 {product}
uintx CMSOldPLABReactivityCeiling = 10 {product}
uintx CMSOldPLABReactivityFactor = 2 {product}
bool CMSOldPLABResizeQuicker = false {product}
uintx CMSOldPLABToleranceFactor = 4 {product}
bool CMSPLABRecordAlways = true {product}
uintx CMSParPromoteBlocksToClaim = 16 {product}
bool CMSParallelRemarkEnabled = true {product}
bool CMSParallelSurvivorRemarkEnabled = true {product}
bool CMSPermGenPrecleaningEnabled = true {product}
uintx CMSPrecleanDenominator = 3 {product}
uintx CMSPrecleanIter = 3 {product}
uintx CMSPrecleanNumerator = 2 {product}
bool CMSPrecleanRefLists1 = true {product}
bool CMSPrecleanRefLists2 = false {product}
bool CMSPrecleanSurvivors1 = false {product}
bool CMSPrecleanSurvivors2 = true {product}
uintx CMSPrecleanThreshold = 1000 {product}
bool CMSPrecleaningEnabled = true {product}
bool CMSPrintChunksInDump = false {product}
```

```
bool CMSPrintObjectsInDump = false {product}
uintx CMSRemarkVerifyVariant = 1 {product}
bool CMSReplenishIntermediate = true {product}
uintx CMSRescanMultiple = 32 {product}
uintx CMSRevisitStackSize = 1048576 {product}
uintx CMSSamplingGrain = 16384 {product}
bool CMSScavengeBeforeRemark = false {product}
uintx CMSScheduleRemarkEdenPenetration = 50 {product}
uintx CMSScheduleRemarkEdenSizeThreshold = 2097152 {product}
uintx CMSScheduleRemarkSamplingRatio = 5 {product}
double CMSSmallCoalSurplusPercent = {product}
double CMSSmallSplitSurplusPercent = {product}
bool CMSSplitIndexedFreeListBlocks = true {product}
intx CMSTriggerPermRatio = 80 {product}
intx CMSTriggerRatio = 80 {product}
bool CMSUseOldDefaults = false {product}
intx CMSWaitDuration = 2000 {product}
uintx CMSWorkQueueDrainThreshold = 10 {product}
bool CMSYield = true {product}
uintx CMSYieldSleepCount = 0 {product}
intx CMSYoungGenPerWorker = 16777216 {product}
uintx CMS_FLSPadding = 1 {product}
uintx CMS_FLSWeight = 75 {product}
uintx CMS_SweepPadding = 1 {product}
uintx CMS_SweepTimerThresholdMillis = 10 {product}
uintx CMS_SweepWeight = 75 {product}
bool CheckJNICalls = false {product}
bool ClassUnloading = true {product}
intx ClearFPUAtPark = 0 {product}
bool ClipInlining = true {product}
```

```
uintx CodeCacheExpansionSize = 32768 {pd product}
uintx CodeCacheFlushingMinimumFreeSpace = 1536000 {product}
uintx CodeCacheMinimumFreeSpace = 512000 {product}
bool CollectGen0First = false {product}
bool CompactFields = true {product}
intx CompilationPolicyChoice = 0 {product}
intx CompilationRepeat = 0 {C1 product}
ccstrlist CompileCommand = {product}
ccstr CompileCommandFile = {product}
ccstrlist CompileOnly = {product}
intx CompileThreshold = 1500 {pd product}
bool CompilerThreadHintNoPreempt = true {product}
intx CompilerThreadPriority = -1 {product}
intx CompilerThreadStackSize = 0 {pd product}
uintx ConcGCThreads = 0 {product}
bool ConvertSleepToYield = true {pd product}
bool ConvertYieldToSleep = false {product}
bool DTraceAllocProbes = false {product}
bool DTraceMethodProbes = false {product}
bool DTraceMonitorProbes = false {product}
uintx DefaultMaxRAMFraction = 4 {product}
intx DefaultThreadPriority = -1 {product}
intx DeferPollingPageLoopCount = -1 {product}
intx DeferThrSuspendLoopCount = 4000 {product}
bool DeoptimizeRandom = false {product}
bool DisableAttachMechanism = false {product}
bool DisableExplicitGC = false {product}
bool DisplayVMOutputToStderr = false {product}
bool DisplayVMOutputToStdout = false {product}
bool DontCompileHugeMethods = true {product}
```

```
bool DontYieldALot = false {pd product}
bool DumpSharedSpaces = false {product}
bool EagerXrunInit = false {product}
intx EmitSync = 0 {product}
uintx ErgoHeapSizeLimit = 0 {product}
ccstr ErrorFile = {product}
bool EstimateArgEscape = true {product}
intx EventLogLength = 2000 {product}
bool ExplicitGCInvokesConcurrent = false {product}
bool ExplicitGCInvokesConcurrentAndUnloadsClasses = false {produ
bool ExtendedDTraceProbes = false {product}
bool FLSAlwaysCoalesceLarge = false {product}
uintx FLSCoalescePolicy = 2 {product}
double FLSLargestBlockCoalesceProximity = {product}
bool FailOverToOldVerifier = true {product}
bool FastTLABRefill = true {product}
intx FenceInstruction = 0 {product}
intx FieldsAllocationStyle = 1 {product}
bool FilterSpuriousWakeups = true {product}
bool ForceFullGCJVMTIEpilogues = false {product}
bool ForceNUMA = false {product}
bool ForceSharedSpaces = false {product}
bool ForceTimeHighResolution = false {product}
intx FreqInlineSize = 325 {pd product}
intx G1ConcRefinementGreenZone = 0 {product}
intx G1ConcRefinementRedZone = 0 {product}
intx G1ConcRefinementServiceIntervalMillis = 300 {product}
uintx G1ConcRefinementThreads = 0 {product}
intx G1ConcRefinementThresholdStep = 0 {product}
intx G1ConcRefinementYellowZone = 0 {product}
```

```
intx G1ConfidencePercent = 50 {product}
uintx G1HeapRegionSize = 0 {product}
intx G1MarkRegionStackSize = 1048576 {product}
intx G1RSetRegionEntries = 0 {product}
uintx G1RSetScanBlockSize = 64 {product}
intx G1RSetSparseRegionEntries = 0 {product}
intx G1RSetUpdatingPauseTimePercent = 10 {product}
intx G1ReservePercent = 10 {product}
intx G1SATBBufferSize = 1024 {product}
intx G1UpdateBufferSize = 256 {product}
bool G1UseAdaptiveConcRefinement = true {product}
bool G1UseFixedWindowMMUTracker = false {product}
uintx GCDrainStackTargetSize = 64 {product}
uintx GCHeapFreeLimit = 2 {product}
bool GCLockerInvokesConcurrent = false {product}
bool GCOverheadReporting = false {product}
intx GCOverheadReportingPeriodMS = 100 {product}
intx GCPauseIntervalMillis = 500 {product}
uintx GCTaskTimeStampEntries = 200 {product}
uintx GCTimeLimit = 98 {product}
uintx GCTimeRatio = 99 {product}
ccstr HPILibPath = {product}
bool HandlePromotionFailure = true {product}
uintx HeapBaseMinAddress = 2147483648 {pd product}
bool HeapDumpAfterFullGC = false {manageable}
bool HeapDumpBeforeFullGC = false {manageable}
bool HeapDumpOnOutOfMemoryError = false {manageable}
ccstr HeapDumpPath = {manageable}
uintx HeapFirstMaximumCompactionCount = 3 {product}
uintx HeapMaximumCompactionInterval = 20 {product}
```

```
bool IgnoreUnrecognizedVMOptions = false {product}
uintx InitialCodeCacheSize = 163840 {pd product}
uintx InitialHeapSize := 16777216 {product}
uintx InitialRAMFraction = 64 {product}
uintx InitialSurvivorRatio = 8 {product}
intx InitialTenuringThreshold = 7 {product}
uintx InitiatingHeapOccupancyPercent = 45 {product}
bool Inline = true {product}
intx InlineSmallCode = 1000 {pd product}
intx InterpreterProfilePercentage = 33 {product}
bool JNIDetachReleasesMonitors = true {product}
bool JavaMonitorsInStackTrace = true {product}
intx JavaPriority10_To_OSPriority = -1 {product}
intx JavaPriority1_To_OSPriority = -1 {product}
intx JavaPriority2_To_OSPriority = -1 {product}
intx JavaPriority3_To_OSPriority = -1 {product}
intx JavaPriority4_To_OSPriority = -1 {product}
intx JavaPriority5_To_OSPriority = -1 {product}
intx JavaPriority6_To_OSPriority = -1 {product}
intx JavaPriority7_To_OSPriority = -1 {product}
intx JavaPriority8_To_OSPriority = -1 {product}
intx JavaPriority9_To_OSPriority = -1 {product}
bool LIRFillDelaySlots = false {C1 pd product}
uintx LargePageHeapSizeThreshold = 134217728 {product}
uintx LargePageSizeInBytes = 0 {product}
bool LazyBootClassLoader = true {product}
bool ManagementServer = false {product}
uintx MarkStackSize = 32768 {product}
uintx MarkStackSizeMax = 4194304 {product}
intx MarkSweepAlwaysCompactCount = 4 {product}
```

```
uintx MarkSweepDeadRatio = 5 {product}
intx MaxBCEAEstimateLevel = 5 {product}
intx MaxBCEAEstimateSize = 150 {product}
intx MaxDirectMemorySize = -1 {product}
bool MaxFDLimit = true {product}
uintx MaxGCMinorPauseMillis = 4294967295 {product}
uintx MaxGCPauseMillis = 4294967295 {product}
uintx MaxHeapFreeRatio = 70 {product}
uintx MaxHeapSize := 268435456 {product}
intx MaxInlineLevel = 9 {product}
intx MaxInlineSize = 35 {product}
intx MaxJavaStackTraceDepth = 1024 {product}
uintx MaxLiveObjectEvacuationRatio = 100 {product}
uintx MaxNewSize = 4294967295 {product}
uintx MaxPermHeapExpansion = 4194304 {product}
uintx MaxPermSize = 67108864 {pd product}
uint64_t MaxRAM = 1073741824 {pd product}
uintx MaxRAMFraction = 4 {product}
intx MaxRecursiveInlineLevel = 1 {product}
intx MaxTenuringThreshold = 15 {product}
intx MaxTrivialSize = 6 {product}
bool MethodFlushing = true {product}
intx MinCodeCacheFlushingInterval = 30 {product}
uintx MinHeapDeltaBytes = 131072 {product}
uintx MinHeapFreeRatio = 40 {product}
intx MinInliningThreshold = 250 {product}
uintx MinPermHeapExpansion = 262144 {product}
uintx MinRAMFraction = 2 {product}
uintx MinSurvivorRatio = 3 {product}
uintx MinTLABSize = 2048 {product}
```

```
intx MonitorBound = 0 {product}
bool MonitorInUseLists = false {product}
bool MustCallLoadClassInternal = false {product}
intx NUMAChunkResizeWeight = 20 {product}
intx NUMAPageScanRate = 256 {product}
intx NUMASpaceResizeRate = 1073741824 {product}
bool NUMAStats = false {product}
intx NativeMonitorFlags = 0 {product}
intx NativeMonitorSpinLimit = 20 {product}
intx NativeMonitorTimeout = -1 {product}
bool NeedsDeoptSuspend = false {pd product}
bool NeverActAsServerClassMachine = true {pd product}
bool NeverTenure = false {product}
intx NewRatio = 2 {product}
uintx NewSize = 1048576 {product}
uintx NewSizeThreadIncrease = 4096 {pd product}
intx NmethodSweepFraction = 4 {product}
uintx OldPLABSize = 1024 {product}
uintx OldPLABWeight = 50 {product}
uintx OldSize = 4194304 {product}
bool OmitStackTraceInFastThrow = true {product}
ccstrlist OnError = {product}
ccstrlist OnOutOfMemoryError = {product}
intx OnStackReplacePercentage = 933 {pd product}
uintx PLABWeight = 75 {product}
bool PSChunkLargeArrays = true {product}
intx ParGCArrayScanChunk = 50 {product}
uintx ParGCDesiredObjsFromOverflowList = 20 {product}
bool ParGCTrimOverflow = true {product}
bool ParGCUseLocalOverflow = false {product}
```

intx ParallelGCBufferWastePct = 10 {product}

bool ParallelGCRetainPLAB = true {product}

uintx ParallelGCThreads = 0 {product}

bool ParallelGCVerbose = false {product}

uintx ParallelOldDeadWoodLimiterMean = 50 {product}

uintx ParallelOldDeadWoodLimiterStdDev = 80 {product}

bool ParallelRefProcBalancingEnabled = true {product}

bool ParallelRefProcEnabled = false {product}

uintx PausePadding = 1 {product}

intx PerBytecodeRecompilationCutoff = 200 {product}

intx PerBytecodeTrapLimit = 4 {product}

intx PerMethodRecompilationCutoff = 400 {product}

intx PerMethodTrapLimit = 100 {product}

bool PerfAllowAtExitRegistration = false {product}

bool PerfBypassFileSystemCheck = false {product}

intx PerfDataMemorySize = 32768 {product}

intx PerfDataSamplingInterval = 50 {product}

ccstr PerfDataSaveFile = {product}

bool PerfDataSaveToFile = false {product}

bool PerfDisableSharedMem = false {product}

intx PerfMaxStringConstLength = 1024 {product}

uintx PermGenPadding = 3 {product}

uintx PermMarkSweepDeadRatio = 20 {product}

uintx PermSize = 12582912 {pd product}

bool PostSpinYield = true {product}

intx PreBlockSpin = 10 {product}

intx PreInflateSpin = 10 {pd product}

bool PreSpinYield = false {product}

bool PreferInterpreterNativeStubs = false {pd product}

intx PrefetchCopyIntervalInBytes = -1 {product}

```
intx PrefetchFieldsAhead = -1 {product}
intx PrefetchScanIntervalInBytes = -1 {product}
bool PreserveAllAnnotations = false {product}
uintx PreserveMarkStackSize = 1024 {product}
uintx PretenureSizeThreshold = 0 {product}
bool PrintAdaptiveSizePolicy = false {product}
bool PrintCMSInitiationStatistics = false {product}
intx PrintCMSStatistics = 0 {product}
bool PrintClassHistogram = false {manageable}
bool PrintClassHistogramAfterFullGC = false {manageable}
bool PrintClassHistogramBeforeFullGC = false {manageable}
bool PrintCommandLineFlags = false {product}
bool PrintCompilation = false {product}
bool PrintConcurrentLocks = false {manageable}
intx PrintFLSCensus = 0 {product}
intx PrintFLSStatistics = 0 {product}
bool PrintFlagsFinal := true {product}
bool PrintFlagsInitial = false {product}
bool PrintGC = false {manageable}
bool PrintGCApplicationConcurrentTime = false {product}
bool PrintGCApplicationStoppedTime = false {product}
bool PrintGCDateStamps = false {manageable}
bool PrintGCDetails = false {manageable}
bool PrintGCTaskTimeStamps = false {product}
bool PrintGCTimeStamps = false {manageable}
bool PrintHeapAtGC = false {product rw}
bool PrintHeapAtGCExtended = false {product rw}
bool PrintHeapAtSIGBREAK = true {product}
bool PrintJNIGCStalls = false {product}
bool PrintJNIResolving = false {product}
```

```
bool PrintOldPLAB = false {product}
bool PrintPLAB = false {product}
bool PrintParallelOldGCPhaseTimes = false {product}
bool PrintPromotionFailure = false {product}
bool PrintReferenceGC = false {product}
bool PrintRevisitStats = false {product}
bool PrintSafepointStatistics = false {product}
intx PrintSafepointStatisticsCount = 300 {product}
intx PrintSafepointStatisticsTimeout = -1 {product}
bool PrintSharedSpaces = false {product}
bool PrintTLAB = false {product}
bool PrintTenuringDistribution = false {product}
bool PrintVMOptions = false {product}
bool PrintVMQWaitTime = false {product}
uintx ProcessDistributionStride = 4 {product}
bool ProfileInterpreter = false {pd product}
bool ProfileIntervals = false {product}
intx ProfileIntervalsTicks = 100 {product}
intx ProfileMaturityPercentage = 20 {product}
bool ProfileVM = false {product}
bool ProfilerPrintByteCodeStatistics = false {product}
bool ProfilerRecordPC = false {product}
uintx PromotedPadding = 3 {product}
intx QueuedAllocationWarningCount = 0 {product}
bool RangeCheckElimination = true {product}
intx ReadPrefetchInstr = 0 {product}
intx ReadSpinIterations = 100 {product}
bool ReduceSignalUsage = false {product}
intx RefDiscoveryPolicy = 0 {product}
bool ReflectionWrapResolutionErrors = true {product}
```

bool RegisterFinalizersAtInit = true {product}

bool RelaxAccessControlCheck = false {product}

bool RequireSharedSpaces = false {product}

uintx ReservedCodeCacheSize = 33554432 {pd product}

bool ResizeOldPLAB = true {product}

bool ResizePLAB = true {product}

bool ResizeTLAB = true {pd product}

bool RestoreMXCSROnJNICalls = false {product}

bool RewriteBytecodes = false {pd product}

bool RewriteFrequentPairs = false {pd product}

intx SafepointPollOffset = 256 {C1 pd product}

intx SafepointSpinBeforeYield = 2000 {product}

bool SafepointTimeout = false {product}

intx SafepointTimeoutDelay = 10000 {product}

bool ScavengeBeforeFullGC = true {product}

intx SelfDestructTimer = 0 {product}

uintx SharedDummyBlockSize = 536870912 {product}

uintx SharedMiscCodeSize = 4194304 {product}

uintx SharedMiscDataSize = 4194304 {product}

uintx SharedReadOnlySize = 10485760 {product}

uintx SharedReadWriteSize = 12582912 {product}

bool ShowMessageBoxOnError = false {product}

intx SoftRefLRUPolicyMSPerMB = 1000 {product}

bool SplitIfBlocks = true {product}

intx StackRedPages = 1 {pd product}

intx StackShadowPages = 3 {pd product}

bool StackTraceInThrowable = true {product}

intx StackYellowPages = 2 {pd product}

bool StartAttachListener = false {product}

intx StarvationMonitorInterval = 200 {product}

```
bool StressLdcRewrite = false {product}
bool StressTieredRuntime = false {product}
bool SuppressFatalErrorMessage = false {product}
uintx SurvivorPadding = 3 {product}
intx SurvivorRatio = 8 {product}
intx SuspendRetryCount = 50 {product}
intx SuspendRetryDelay = 5 {product}
intx SyncFlags = 0 {product}
ccstr SyncKnobs = {product}
intx SyncVerbose = 0 {product}
uintx TLABAllocationWeight = 35 {product}
uintx TLABRefillWasteFraction = 64 {product}
uintx TLABSize = 0 {product}
bool TLABStats = true {product}
uintx TLABWasteIncrement = 4 {product}
uintx TLABWasteTargetPercent = 1 {product}
bool TaggedStackInterpreter = false {product}
intx TargetPLABWastePct = 10 {product}
intx TargetSurvivorRatio = 50 {product}
uintx TenuredGenerationSizeIncrement = 20 {product}
uintx TenuredGenerationSizeSupplement = 80 {product}
uintx TenuredGenerationSizeSupplementDecay = 2 {product}
intx ThreadPriorityPolicy = 0 {product}
bool ThreadPriorityVerbose = false {product}
uintx ThreadSafetyMargin = 52428800 {product}
intx ThreadStackSize = 0 {pd product}
uintx ThresholdTolerance = 10 {product}
intx Tier1BytecodeLimit = 10 {product}
bool Tier1OptimizeVirtualCallProfiling = true {C1 product}
bool Tier1ProfileBranches = true {C1 product}
```

```
bool Tier1ProfileCalls = true {C1 product}
bool Tier1ProfileCheckcasts = true {C1 product}
bool Tier1ProfileInlinedCalls = true {C1 product}
bool Tier1ProfileVirtualCalls = true {C1 product}
bool Tier1UpdateMethodData = false {product}
intx Tier2BackEdgeThreshold = 100000 {pd product}
intx Tier2CompileThreshold = 1500 {pd product}
intx Tier3BackEdgeThreshold = 100000 {pd product}
intx Tier3CompileThreshold = 2500 {pd product}
intx Tier4BackEdgeThreshold = 100000 {pd product}
intx Tier4CompileThreshold = 4500 {pd product}
bool TieredCompilation = false {pd product}
bool TimeLinearScan = false {C1 product}
bool TraceBiasedLocking = false {product}
bool TraceClassLoading = false {product rw}
bool TraceClassLoadingPreorder = false {product}
bool TraceClassResolution = false {product}
bool TraceClassUnloading = false {product rw}
bool TraceGen0Time = false {product}
bool TraceGen1Time = false {product}
ccstr TraceJVMTI = {product}
bool TraceLoaderConstraints = false {product rw}
bool TraceMonitorInflation = false {product}
bool TraceParallelOldGCTasks = false {product}
intx TraceRedefineClasses = 0 {product}
bool TraceSafepointCleanupTime = false {product}
bool TraceSuspendWaitFailures = false {product}
intx TypeProfileMajorReceiverPercent = 90 {product}
intx TypeProfileWidth = 2 {product}
intx UnguardOnExecutionViolation = 0 {product}
```

```
bool Use486InstrsOnly = false {product}
bool UseAdaptiveGCBoundary = false {product}
bool UseAdaptiveGenerationSizePolicyAtMajorCollection = true {p
bool UseAdaptiveGenerationSizePolicyAtMinorCollection = true {p
bool UseAdaptiveNUMAChunkSizing = true {product}
bool UseAdaptiveSizeDecayMajorGCCost = true {product}
bool UseAdaptiveSizePolicy = true {product}
bool UseAdaptiveSizePolicyFootprintGoal = true {product}
bool UseAdaptiveSizePolicyWithSystemGC = false {product}
bool UseAddressNop = false {product}
bool UseAltSigs = false {product}
bool UseAutoGCSelectPolicy = false {product}
bool UseBiasedLocking = true {product}
bool UseBoundThreads = true {product}
bool UseCMSBestFit = true {product}
bool UseCMSCollectionPassing = true {product}
bool UseCMSCompactAtFullCollection = true {product}
bool UseCMSInitiatingOccupancyOnly = false {product}
bool UseCodeCacheFlushing = false {product}
bool UseCompiler = true {product}
bool UseCompilerSafepoints = true {product}
bool UseConcMarkSweepGC = false {product}
bool UseCountLeadingZerosInstruction = false {product}
bool UseCounterDecay = true {product}
bool UseDepthFirstScavengeOrder = true {product}
bool UseFastAccessorMethods = true {product}
bool UseFastEmptyMethods = true {product}
bool UseFastJNIAccessors = true {product}
bool UseG1GC = false {product}
bool UseGCOverheadLimit = true {product}
```

```
bool UseGCTaskAffinity = false {product}
bool UseHeavyMonitors = false {product}
bool UseInlineCaches = true {product}
bool UseInterpreter = true {product}
bool UseLWPSynchronization = true {product}
bool UseLargePages = false {pd product}
bool UseLargePagesIndividualAllocation := false {pd product}
bool UseLoopCounter = true {product}
bool UseMaximumCompactionOnSystemGC = true {product}
bool UseMembar = false {product}
bool UseNUMA = false {product}
bool UseNewFeature1 = false {C1 product}
bool UseNewFeature2 = false {C1 product}
bool UseNewFeature3 = false {C1 product}
bool UseNewFeature4 = false {C1 product}
bool UseNewLongLShift = false {product}
bool UseNiagaraInstrs = false {product}
bool UseOSErrorReporting = false {pd product}
bool UseOnStackReplacement = true {pd product}
bool UsePSAdaptiveSurvivorSizePolicy = true {product}
bool UseParNewGC = false {product}
bool UseParallelDensePrefixUpdate = true {product}
bool UseParallelGC = false {product}
bool UseParallelOldGC = false {product}
bool UseParallelOldGCCompacting = true {product}
bool UseParallelOldGCDensePrefix = true {product}
bool UsePerfData = true {product}
bool UsePopCountInstruction = false {product}
intx UseSSE = 99 {product}
bool UseSSE42Intrinsics = false {product}
```

```
bool UseSerialGC = false {product}
bool UseSharedSpaces = true {product}
bool UseSignalChaining = true {product}
bool UseSpinning = false {product}
bool UseSplitVerifier = true {product}
bool UseStoreImmI16 = true {product}
bool UseStringCache = false {product}
bool UseTLAB = true {pd product}
bool UseThreadPriorities = true {pd product}
bool UseTypeProfile = true {product}
bool UseUTCFileTimestamp = true {product}
bool UseUnalignedLoadStores = false {product}
bool UseVMInterruptibleIO = true {product}
bool UseVectoredExceptions = false {pd product}
bool UseXMMForArrayCopy = false {product}
bool UseXmmI2D = false {product}
bool UseXmmI2F = false {product}
bool UseXmmLoadAndClearUpper = true {product}
bool UseXmmRegToRegMoveAll = false {product}
bool VMThreadHintNoPreempt = false {product}
intx VMThreadPriority = -1 {product}
intx VMThreadStackSize = 0 {pd product}
intx ValueMapInitialSize = 11 {C1 product}
intx ValueMapMaxLoopSize = 8 {C1 product}
bool VerifyMergedCPBytecodes = true {product}
intx WorkAroundNPTLTimedWaitHang = 1 {product}
uintx YoungGenerationSizeIncrement = 20 {product}
uintx YoungGenerationSizeSupplement = 80 {product}
uintx YoungGenerationSizeSupplementDecay = 8 {product}
uintx YoungPLABSize = 4096 {product}
```

```
bool ZeroTLAB = false {product}
intx hashCode = 0 {product}
```

来源： http://blog.csdn.net/bbaiggey/article/details/50885943