<https://www.cnblogs.com/xifenglou/p/9045447.html>

========================================================================================================================================================================================================================================================================

java基础强化深入学习java教程之jvm垃圾回收机制

<https://www.bilibili.com/video/av94243744?p=1>

29节 讲到了jprofiler

IT黑马

========================================================================================================================================================================================================================================================================

-Xms10M

-Xmx10M

-XX:+PrintGCDetails

-XX:+PrintGCTimeStamps

-XX:+PrintGCDateStamps

-XX:+HeapDumpOnOutOfMemoryError

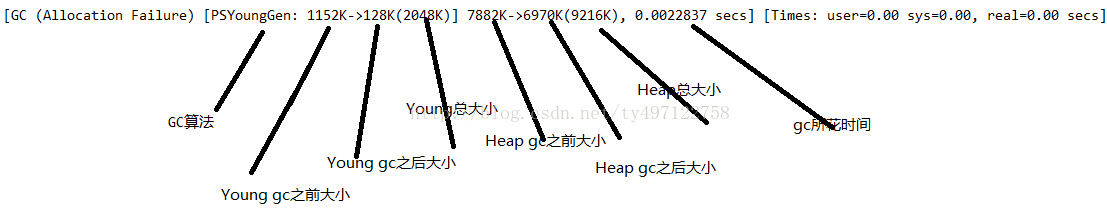
-XX:HeapDumpPath=d:\oomdump.dump

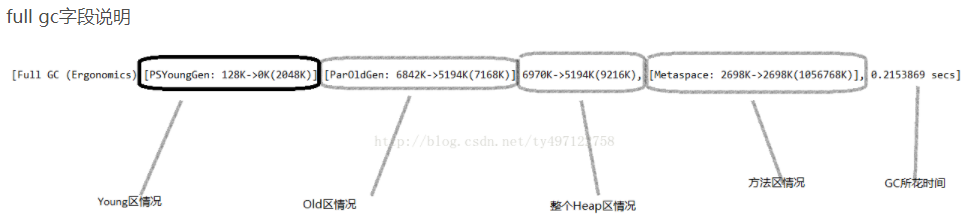
-XX:+PrintHeapAtGC

*-verbose:gc -Xms20M -Xmx20M -Xmn10M -XX:+PrintGCDateStamps -XX:+PrintGCDetails -XX:SurvivorRatio=8 -XX:+PrintHeapAtGC*

[GC (Allocation Failure) [PSYoungGen: 2048K->496K(2560K)] 2048K->792K(9728K), 0.0008979 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]

<https://blog.csdn.net/ty497122758/article/details/74231560>





串行收集器：

DefNew：是使用-XX:+UseSerialGC（新生代，老年代都使用串行回收收集器）。

并行收集器：

ParNew：是使用-XX:+UseParNewGC（新生代使用并行收集器，老年代使用串行回收收集器）或者-XX:+UseConcMarkSweepGC(新生代使用并行收集器，老年代使用CMS)。

PSYoungGen：是使用-XX:+UseParallelOldGC（新生代，老年代都使用并行回收收集器）或者-XX:+UseParallelGC（新生代使用并行回收收集器，老年代使用串行收集器）

garbage-first heap：是使用-XX:+UseG1GC（G1收集器）

————————————————

原文链接：https://blog.csdn.net/ty497122758/article/details/74231560

一、Minor GC触发条件

1、eden区满时，触发MinorGC。即申请一个对象时，发现eden区不够用，则触发一次MinorGC。

注：新生代分为三个区域，eden space, from space, to space。默认比例是8:1:1。在MinorGC时，会把存活的对象复制到to space区域，如果to space区域不够，则利用担保机制进入老年代区域。

对eden space, from space, to space的理解：每次分配eden space空间，如果不够，则小于 to space大小的对象复制到 to space，然后to space和from space换位置，所以我们看到的to space一直是空的。

————————————————

原文链接：<https://blog.csdn.net/shi2huang/article/details/80067608>

JVM内存模型和垃圾回收机制

<https://blog.csdn.net/y277an/article/details/93815647>

================================================================================================================================================================================

========================================================================================

\*\*\*\*\*\*\*写的非常好，必须仔细阅读5遍以上\*\*\*\*\*\*\*

========================================================================================

GC:

https://blog.csdn.net/weixin\_34109408/article/details/85748049

原创：https://www.cnblogs.com/qianguyihao/p/4744233.html

GC：Garbage Collection 垃圾收集

1960年 Lisp使用了GC

Java中，GC的对象是Java堆和方法区（即永久区）

五、复制算法：（新生代的GC）

现在的商业虚拟机都采用这种收集算法来回收新生代，新生代中的对象98%都是“朝生夕死”的，所以并不需要按照1:1的比例来划分内存空间，而是将内存分为一块比较大的Eden空间和两块较小的Survivor空间，每次使用Eden和其中一块Survivor。当回收时，将Eden和Survivor中还存活着的对象一次性地复制到另外一块Survivor空间上，最后清理掉Eden和刚才用过的Survivor空间。HotSpot虚拟机默认Eden和Survivor的大小比例是8:1，也就是说，每次新生代中可用内存空间为整个新生代容量的90%（80%+10%），只有10%的空间会被浪费。

当然，98%的对象可回收只是一般场景下的数据，我们没有办法保证每次回收都只有不多于10%的对象存活，当Survivor空间不够用时，需要依赖于老年代进行分配担保，所以大对象直接进入老年代。

================================================================================================================================================================================

========================================================================================

-XX:+PrintGC 输出简要GC日志

-XX:+PrintGCDetails 输出详细GC日志

-Xloggc:gc.log 输出GC日志到文件

-XX:+PrintGCTimeStamps 输出GC的时间戳（以JVM启动到当期的总时长的时间戳形式）

-XX:+PrintGCDateStamps 输出GC的时间戳（以日期的形式，如 2013-05-04T21:53:59.234+0800）

-XX:+PrintHeapAtGC 在进行GC的前后打印出堆的信息

-verbose:gc

-XX:+PrintReferenceGC 打印年轻代各个引用的数量以及时长

作者：青1475864380702

链接：https://juejin.im/post/5c80b0f451882532cd57b541

来源：掘金

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

========================================================================================

================================================================================================================================================================================

<http://www.51gjie.com/java/551.html>

参数名称 含义 默认值

-Xms 初始堆大小 物理内存的1/64(<1GB) 默认(MinHeapFreeRatio参数可以调整)空余堆内存小于40%时，JVM就会增大堆直到-Xmx的最大限制.

-Xmx 最大堆大小 物理内存的1/4(<1GB) 默认(MaxHeapFreeRatio参数可以调整)空余堆内存大于70%时，JVM会减少堆直到 -Xms的最小限制

-Xmn 年轻代大小(1.4or lator) 注意：此处的大小是（eden+ 2 survivor space).与jmap -heap中显示的New gen是不同的。

整个堆大小=年轻代大小 + 年老代大小 + 持久代大小.

增大年轻代后,将会减小年老代大小.此值对系统性能影响较大,Sun官方推荐配置为整个堆的3/8

========================================================================================================================================================================================================================================================================

<https://www.ituring.com.cn/article/195798>

Java虚拟机的内存分配，大体上可分为年轻代（Young Generation），老年代（Tenured Generation）以及元空间（Metaspace）。其中，新生代又分为Eden（伊甸园）,S0和S1（Survivor，幸存者乐园）。（参考《Java程序员修炼之道》及VisualVM）

注：Java 8取消了原来的永久代（PermGen），改为元空间（Metaspace）了。

对应于Java虚拟机的参数设置：

-Xmn的大小 = Eden + S0 + S1

-Xmx的大小 = Eden + S0 + S1 + Old

-XX:SurvivorRatio用来分配Eden和S0，S1的比例

-XX:MaxMetaspaceSize表示Metaspace的最大值（-XX:MaxPermSize在Java 8中已经没用了）

========================================================================================================================================================================================================================================================================

<https://hllvm-group.iteye.com/group/topic/37095>

串行收集器：

DefNew：是使用-XX:+UseSerialGC（新生代，老年代都使用串行回收收集器）。

并行收集器：

ParNew：是使用-XX:+UseParNewGC（新生代使用并行收集器，老年代使用串行回收收集器）或者-XX:+UseConcMarkSweepGC(新生代使用并行收集器，老年代使用CMS)。

PSYoungGen：是使用-XX:+UseParallelOldGC（新生代，老年代都使用并行回收收集器）或者-XX:+UseParallelGC（新生代使用并行回收收集器，老年代使用串行收集器）

garbage-first heap：是使用-XX:+UseG1GC（G1收集器）

================================================================================================================================================================================

========================================================================================

<https://xbuba.com/questions/47192751>

在JDK 7u40中，并行垃圾收集器的年轻代的最小大小在32位JVM中从192 KB增加到768 KB，在64位JVM中增加到1536 KB。JVM无法正确检查此新的最小大小。如果在命令行上指定了小于新的最小值的年轻代大小，则可能导致崩溃或性能下降。

年轻一代的大小由选项-XX：NewSize =和-XX：MaxNewSize =设置，或由选项-Xmn设置（后一个选项等效于将NewSize和MaxNewSize都设置为）。如果未使用上述选项，则将年轻代大小计算为最大堆大小的一部分。

解决方法：使用最小的新一代大小至少为768 KB（对于32位JVM）或1536 KB（对于64位JVM）。

================================================================================================================================================================================

========================================================================================

<https://blog.csdn.net/someby/article/details/83713476>

Heap

 PSYoungGen      total 2560K, used 2505K [0x00000000ffd00000, 0x0000000100000000, 0x0000000100000000)

  eden space 2048K, 98% used [0x00000000ffd00000,0x00000000ffef64d8,0x00000000fff00000)

  from space 512K, 97% used [0x00000000fff00000,0x00000000fff7c2f0,0x00000000fff80000)

  to   space 512K, 0% used [0x00000000fff80000,0x00000000fff80000,0x0000000100000000)

 ParOldGen       total 7168K, used 1196K [0x00000000ff600000, 0x00000000ffd00000, 0x00000000ffd00000)

  object space 7168K, 16% used [0x00000000ff600000,0x00000000ff72b180,0x00000000ffd00000)

 Metaspace       used 3377K, capacity 4496K, committed 4864K, reserved 1056768K

  class space    used 352K, capacity 388K, committed 512K, reserved 1048576K

PSYoungGen:GC日志中的PSYoungGen（PS是指Parallel Scavenge）为Eden+FromSpace，而整个YoungGeneration为Eden+FromSpace+ToSpace。

    ParOldGen:ParOldGen表示gc回收前后老年代的内存变化

    MetaSpace:JDK1.7中，存储在永久代的部分数据就已经转移到了Java Heap或者是 Native Heap。但永久代仍存在于JDK1.7中，并没完全移除，譬如符号引用(Symbols)转移到了native heap；字面量(interned strings)转移到了java heap；类的静态变量(class statics)转移到了java heap。

Parallel Scavenge 的介绍参考 https://hllvm-group.iteye.com/group/topic/37095

================================================================================================================================================================================

========================================================================================