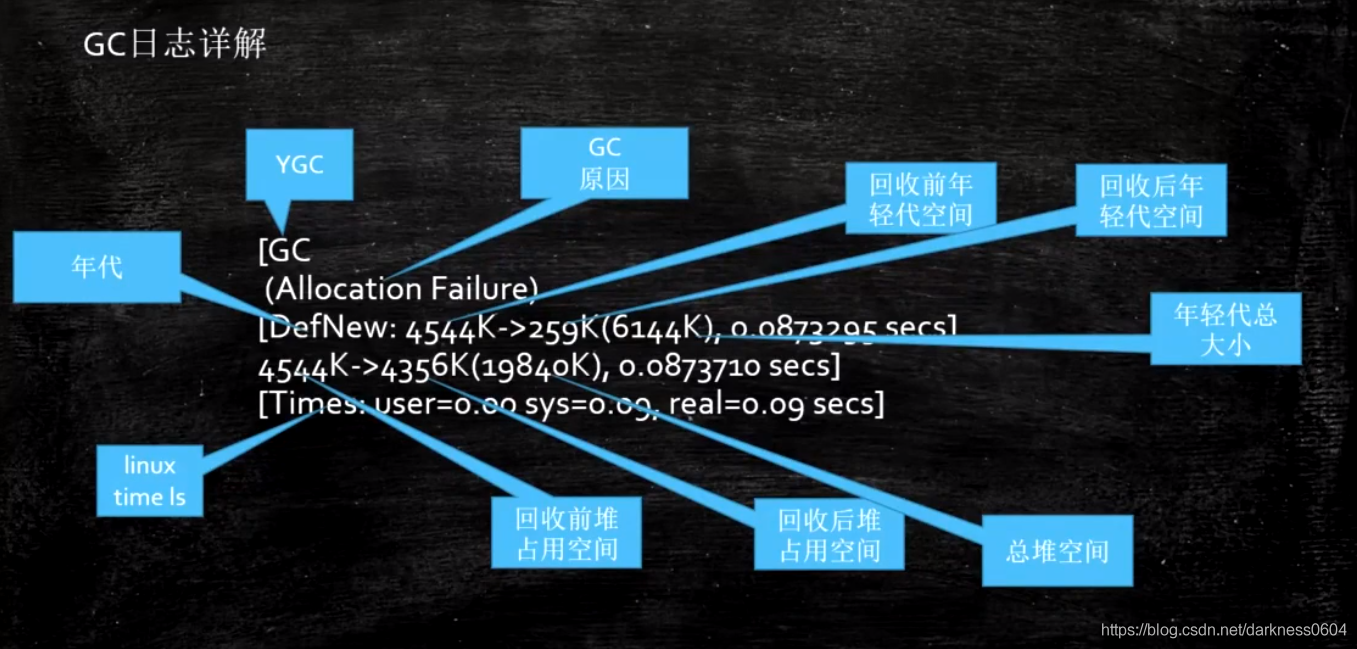
每种垃圾回收器的格式都是不同的，这里我们分别分析一下ps，cms，g1的日志。

**Parallel Scavenge日志**

|  |
| --- |
| [root@dream01 jvm]# java -XX:+PrintGCDetails HelloGC  HelloGC!  //第一段  [GC (Allocation Failure) [DefNew: 7675K->256K(9216K), 0.0040152 secs] 7675K->7424K(29696K), 0.0040475 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]  [GC (Allocation Failure) [DefNew: 7586K->255K(9216K), 0.0023575 secs] 14754K->14592K(29696K), 0.0023767 secs] [Times: user=0.01 sys=0.00, real=0.01 secs]  [GC (Allocation Failure) [DefNew: 7577K->255K(9216K), 0.0060836 secs][Tenured: 21504K->21507K(21508K), 0.0029740 secs] 21913K->21759K(30724K), [Metaspace: 2484K->2484K(1056768K)], 0.0091329 secs] [Times: user=0.00 sys=0.01, real=0.01 secs]  [GC (Allocation Failure) [DefNew: 13837K->1275K(16192K), 0.0046927 secs] 35345K->35071K(52040K), 0.0047138 secs] [Times: user=0.01 sys=0.00, real=0.00 secs]  [GC (Allocation Failure) [DefNew: 14863K->1024K(16192K), 0.0088776 secs][Tenured: 47359K->47360K(48184K), 0.0020449 secs] 48659K->48384K(64376K), [Metaspace: 2484K->2484K(1056768K)], 0.0110120 secs] [Times: user=0.01 sys=0.00, real=0.02 secs]  [GC (Allocation Failure) [DefNew: 31331K->3072K(35584K), 0.0104737 secs] 78691K->78081K(114520K), 0.0104986 secs] [Times: user=0.01 sys=0.01, real=0.01 secs]  [GC (Allocation Failure) [DefNew: 34400K->3072K(35584K), 0.0168687 secs][Tenured: 105729K->105729K(106692K), 0.0045656 secs] 109409K->108802K(142276K), [Metaspace: 2484K->2484K(1056768K)], 0.0215691 secs] [Times: user=0.01 sys=0.01, real=0.02 secs]  [GC (Allocation Failure) [DefNew: 69973K->8194K(79360K), 0.0274334 secs] 175703K->175365K(255580K), 0.0274600 secs] [Times: user=0.02 sys=0.01, real=0.03 secs]  [GC (Allocation Failure) [DefNew: 78152K->8193K(79360K), 0.0374659 secs][Tenured: 236806K->236806K(236872K), 0.0031278 secs] 245323K->244999K(316232K), [Metaspace: 2484K->2484K(1056768K)], 0.0407589 secs] [Times: user=0.02 sys=0.03, real=0.04 secs]  [GC (Allocation Failure) [DefNew (promotion failed) : 126376K->141740K(143168K), 0.0353498 secs][Tenured: 317703K->317703K(318144K), 0.0122563 secs] 363182K->362764K(461312K), [Metaspace: 2485K->2485K(1056768K)], 0.0477168 secs] [Times: user=0.03 sys=0.02, real=0.05 secs]  [Full GC (Allocation Failure) [Tenured: 317703K->317703K(318144K), 0.0197089 secs] 459401K->459023K(461312K), [Metaspace: 2485K->2485K(1056768K)], 0.0197452 secs] [Times: user=0.02 sys=0.00, real=0.02 secs]  [Full GC (Allocation Failure) [Tenured: 317703K->317703K(318144K), 0.0023578 secs] 460359K->460048K(461312K), [Metaspace: 2485K->2485K(1056768K)], 0.0023845 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]  [Full GC (Allocation Failure) [Tenured: 317703K->317692K(318144K), 0.0386269 secs] 460048K->460036K(461312K), [Metaspace: 2485K->2485K(1056768K)], 0.0386466 secs] [Times: user=0.04 sys=0.00, real=0.04 secs]  Exception in thread "main" java.lang.OutOfMemoryError: Java heap space  at HelloGC.main(HelloGC.java:8)  //第二段  Heap  def new generation total 143168K, used 142657K [0x00000000e2e00000, 0x00000000ec950000, 0x00000000ec950000)  eden space 127296K, 100% used [0x00000000e2e00000, 0x00000000eaa50000, 0x00000000eaa50000)  from space 15872K, 96% used [0x00000000eb9d0000, 0x00000000ec8d0478, 0x00000000ec950000)  to space 15872K, 0% used [0x00000000eaa50000, 0x00000000eaa50000, 0x00000000eb9d0000)  tenured generation total 318144K, used 317692K [0x00000000ec950000, 0x0000000100000000, 0x0000000100000000)  the space 318144K, 99% used [0x00000000ec950000, 0x00000000fff8f008, 0x00000000fff8f200, 0x0000000100000000)  Metaspace used 2516K, capacity 4486K, committed 4864K, reserved 1056768K  class space used 272K, capacity 386K, committed 512K, reserved 1048576K |

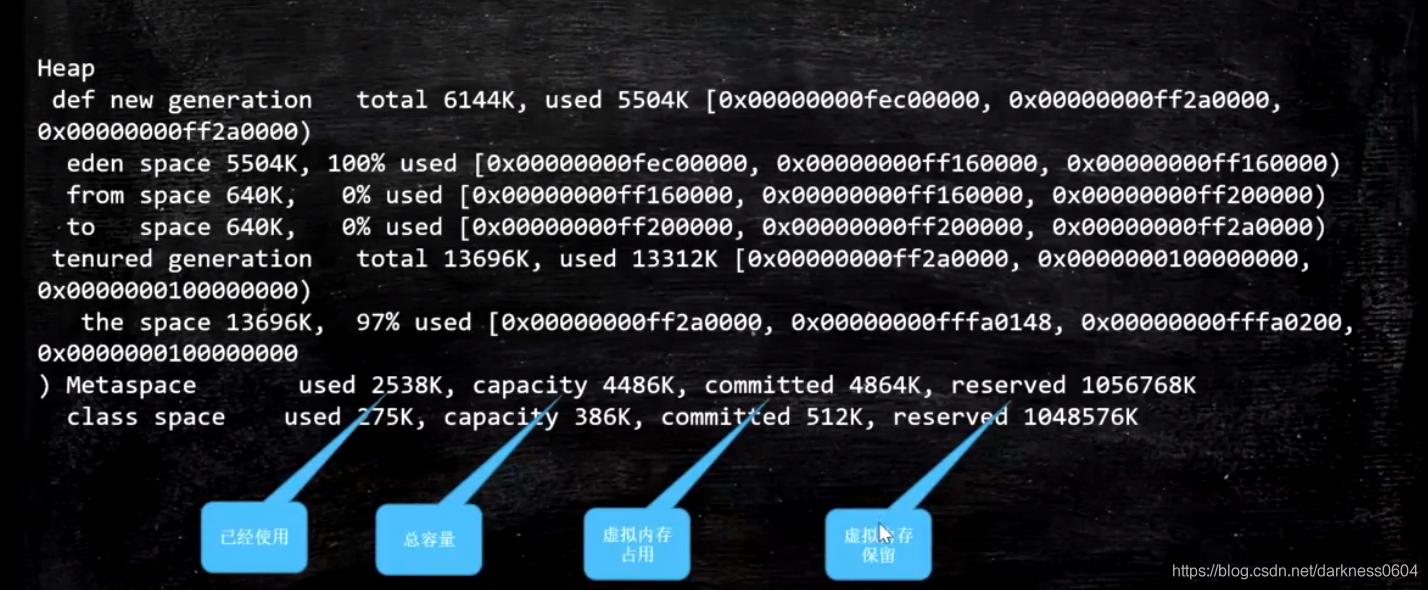
我们打印GC日志，可以发现日志分为两段内容：

第一段分析：



第一段主要是GC回收情况的概览。

第二段：



total = eden + 1个survivor

上面就是新生代，老年代的信息，随便拿一段：

404224101385.png

可以看到这里的每一段描述后面都跟了三个地址。都代表什么意思呢？

后面的内存地址依次指的是，起始地址，使用空间结束地址，整体空间结束地址

在向下看：

404224107311.png

可以看到metaspace，也就是元空间的占用情况，以及在它里面关于存放class信息的那块空间的存放信息。

可以发现它们两个都包含了四个值：

reserved（虚拟内存保留）：是说操作系统给它预留虚拟内存的总大小

committed（虚拟内存占用）：操作系统是以页为单位进行占用内存的，因此一般都会比申请的总容量大小多出一些大小。

capacity（总容量大小）：就是实际指定申请的总容量大小，虚拟内存占用一般都会比它多出一些大小。

used（已经使用）：已经使用了的大小。

**CMS日志**

1、执行命令：

|  |
| --- |
| java -Xms20M -Xmx20M -XX:+PrintGCDetails -XX:+UseConcMarkSweepGC com.peng.jvm.gc.T15\_FullGC\_Problem01 |

首先触发YGC，打印日志:

|  |
| --- |
| [GC (Allocation Failure) [ParNew: 6144K->640K(6144K), 0.0265885 secs] 6585K->2770K(19840K), 0.0268035 secs] [Times: user=0.02 sys=0.00, real=0.02 secs]  ParNew：年轻代收集器  6144->640：收集前后的占用空间对比  （6144）：整个年轻代容量  6585 -> 2770：收集前后的整个堆的占用空间情况对比  （19840）：整个堆大小 |

当若干次YGC后，达到了Old区的CMS清理阈值，触发CMS清理：

|  |
| --- |
| [GC (CMS Initial Mark) [1 CMS-initial-mark: 8511K(13696K)] 9866K(19840K), 0.0040321 secs] [Times: user=0.01 sys=0.00, real=0.00 secs]  //8511 (13696) : 老年代使用空间（老年代最大空间）  //9866 (19840) : 整个堆使用空间（整个堆最大空间）  [CMS-concurrent-mark-start]  [CMS-concurrent-mark: 0.018/0.018 secs] [Times: user=0.01 sys=0.00, real=0.02 secs]  //这里的时间意义不大，因为是并发执行  [CMS-concurrent-preclean-start]  [CMS-concurrent-preclean: 0.000/0.000 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]  //preclean阶段，标记Card为Dirty，也称为Card Marking  [GC (CMS Final Remark) [YG occupancy: 1597 K (6144 K)][Rescan (parallel) , 0.0008396 secs][weak refs processing, 0.0000138 secs][class unloading, 0.0005404 secs][scrub symbol table, 0.0006169 secs][scrub string table, 0.0004903 secs][1 CMS-remark: 8511K(13696K)] 10108K(19840K), 0.0039567 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]  //STW阶段，YG occupancy:年轻代占用及容量  //[Rescan (parallel)：STW下的存活对象标记  //weak refs processing: 弱引用处理  //class unloading: 卸载用不到的class  //scrub symbol(string) table: （清理包含类级元数据和内部字符串的符号表和字符串表，将卸载的class的元数据清洗掉）  //cleaning up symbol and string tables which hold class-level metadata and  //internalized string respectively  //CMS-remark: 8511K(13696K): remark阶段过后的老年代占用及容量  //10108K(19840K): remark阶段过后的堆占用及容量  [CMS-concurrent-sweep-start]  [CMS-concurrent-sweep: 0.005/0.005 secs] [Times: user=0.00 sys=0.00, real=0.01 secs]  //标记已经完成，进行并发清理  [CMS-concurrent-reset-start]  [CMS-concurrent-reset: 0.000/0.000 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]  //重置内部结构（将card表的dirty位重置），为下次GC做准备 |

**G1日志**

1、输出命令“-XX:+UseG1GC”指定G1垃圾收集器

|  |
| --- |
| java -Xms20M -Xmx20M -XX:+PrintGCDetails -XX:+UseG1GC com.peng.jvm.gc.T15\_FullGC\_Problem01 |

G1的YGC和混合GC是可以同时进行的。

|  |
| --- |
| [GC pause (G1 Evacuation Pause) (young) (initial-mark), 0.0015790 secs]  //young -> 年轻代 Evacuation-> 复制存活对象  //initial-mark 混合回收的阶段，这里是YGC混合老年代回收  [Parallel Time: 1.5 ms, GC Workers: 1] //一个GC线程  [GC Worker Start (ms): 92635.7]  [Ext Root Scanning (ms): 1.1]  [Update RS (ms): 0.0] //更新RSet  [Processed Buffers: 1]  [Scan RS (ms): 0.0] //扫描Rset  [Code Root Scanning (ms): 0.0]  [Object Copy (ms): 0.1]  [Termination (ms): 0.0]  [Termination Attempts: 1]  [GC Worker Other (ms): 0.0]  [GC Worker Total (ms): 1.2]  [GC Worker End (ms): 92636.9]  [Code Root Fixup: 0.0 ms]  [Code Root Purge: 0.0 ms]  [Clear CT: 0.0 ms]  [Other: 0.1 ms]  [Choose CSet: 0.0 ms]  [Ref Proc: 0.0 ms]  [Ref Enq: 0.0 ms]  [Redirty Cards: 0.0 ms]  [Humongous Register: 0.0 ms]  [Humongous Reclaim: 0.0 ms]  [Free CSet: 0.0 ms] //释放CollectionSet  [Eden: 0.0B(1024.0K)->0.0B(1024.0K) Survivors: 0.0B->0.0B Heap: 18.8M(20.0M)->18.8M(20.0M)] //YGC后的内存占用情况  [Times: user=0.00 sys=0.00, real=0.00 secs]  //以下是混合回收其他阶段  [GC concurrent-root-region-scan-start]  [GC concurrent-root-region-scan-end, 0.0000078 secs]  [GC concurrent-mark-start]  //无法evacuation，进行FGC  [Full GC (Allocation Failure) 18M->18M(20M), 0.0719656 secs]  [Eden: 0.0B(1024.0K)->0.0B(1024.0K) Survivors: 0.0B->0.0B Heap: 18.8M(20.0M)->18.8M(20.0M)], [Metaspace: 38  76K->3876K(1056768K)] [Times: user=0.07 sys=0.00, real=0.07 secs] |