**快速解读GC日志**

<https://blog.csdn.net/renfufei/article/details/49230943>

本文是 Plumbr 发行的 [Java垃圾收集指南](https://blog.csdn.net/renfufei/column/info/14851) 的部分内容。文中将介绍GC日志的输出格式, 以及如何解读GC日志, 从中提取有用的信息。我们通过 -XX:+UseSerialGC 选项,指定JVM使用串行垃圾收集器, 并使用下面的启动参数让 JVM 打印出详细的GC日志:

-XX:+PrintGCDetails

-XX:+PrintGCDateStamps

-XX:+PrintGCTimeStamps

* 1
* 2
* 3

这样配置以后，发生GC时输出的日志就类似于下面这种格式(为了显示方便,已手工折行):

2015-05-26T14:45:37.987-0200: 151.126:

[GC (Allocation Failure) 151.126:

[DefNew: 629119K->69888K(629120K), 0.0584157 secs]

1619346K->1273247K(2027264K), 0.0585007 secs]

[Times: user=0.06 sys=0.00, real=0.06 secs]

2015-05-26T14:45:59.690-0200: 172.829:

[GC (Allocation Failure) 172.829:

[DefNew: 629120K->629120K(629120K), 0.0000372 secs]

172.829: [Tenured: 1203359K->755802K(1398144K), 0.1855567 secs]

1832479K->755802K(2027264K),

[Metaspace: 6741K->6741K(1056768K)], 0.1856954 secs]

[Times: user=0.18 sys=0.00, real=0.18 secs]

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13

上面的GC日志暴露了JVM中的一些信息。事实上，这个日志片段中发生了 **2** 次垃圾回收事件(Garbage Collection events)。其中一次清理的是年轻代(Young generation), 而第二次处理的是整个堆内存。下面我们来看，如何解读第一次GC事件，发生在年轻代中的小型GC(Minor GC):

2015-05-26T14:45:37.987-02001:151.1262:  
  
[GC3(Allocation Failure4)151.126:   
  
[DefNew5:629119K->69888K6(629120K)7, 0.0584157 secs]  
1619346K->1273247K8(2027264K)9,0.0585007 secs10]

[Times: user=0.06 sys=0.00, real=0.06 secs]11

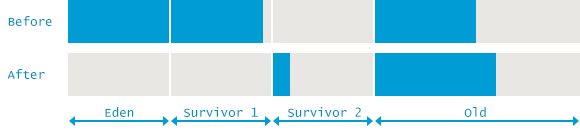
1. 2015-05-26T14:45:37.987-0200 – GC事件(GC event)开始的时间点.
2. 151.126 – GC事件的开始时间,相对于JVM的启动时间,单位是秒(Measured in seconds).
3. GC – 用来区分(distinguish)是 Minor GC 还是 Full GC 的标志(Flag). 这里的 GC 表明本次发生的是 Minor GC.
4. Allocation Failure – 引起垃圾回收的原因. 本次GC是因为年轻代中没有任何合适的区域能够存放需要分配的数据结构而触发的.
5. DefNew – 使用的垃圾收集器的名字. DefNew 这个名字代表的是: 单线程(single-threaded), 采用标记复制(mark-copy)算法的, 使整个JVM暂停运行(stop-the-world)的年轻代(Young generation) 垃圾收集器(garbage collector).
6. 629119K->69888K – 在本次垃圾收集之前和之后的年轻代内存使用情况(Usage).
7. (629120K) – 年轻代的总的大小(Total size).
8. 1619346K->1273247K – 在本次垃圾收集之前和之后整个堆内存的使用情况(Total used heap).
9. (2027264K) – 总的可用的堆内存(Total available heap).
10. 0.0585007 secs – GC事件的持续时间(Duration),单位是秒.
11. [Times: user=0.06 sys=0.00, real=0.06 secs] – GC事件的持续时间,通过多种分类来进行衡量:

* user – 此次垃圾回收, 垃圾收集线程消耗的所有CPU时间(Total CPU time).
* sys – 操作系统调用(OS call) 以及等待系统事件的时间(waiting for system event)
* real – 应用程序暂停的时间(Clock time). 由于串行垃圾收集器(Serial Garbage Collector)只会使用单个线程, 所以 real time 等于 user 以及 system time 的总和.

通过上面的分析, 我们可以计算出在垃圾收集期间, JVM 中的内存使用情况。在垃圾收集之前, 堆内存总的使用了 **1.54G**(1,619,346K)。其中, 年轻代使用了 **614M**(629,119k)。可以算出老年代使用的内存为: **967M**(990,227K)。

下一组数据( -> 右边)中蕴含了更重要的结论, 年轻代的内存使用在垃圾回收后下降了 **546M**(559,231k), 但总的堆内存使用(total heap usage)只减少了 **337M**(346,099k). 通过这一点,我们可以计算出, 有 **208M**(213,132K) 的年轻代对象被提升到老年代(Old)中。

这个GC事件可以用下面的示意图来表示, 上方表示GC之前的内存使用情况, 下方表示结束后的内存使用情况:



如果你想学习更多, 请查看完整的 [Java垃圾收集指南](http://blog.csdn.net/column/details/14851.html), 本示例是从其中抽取的。

原文链接: [Understanding Garbage Collection Logs](https://plumbr.eu/blog/garbage-collection/understanding-garbage-collection-logs)

翻译日期: 2015年10月18日

翻译人员: [铁锚 http://blog.csdn.net/renfufei](http://blog.csdn.net/renfufei)

**百度经验：**

<https://jingyan.baidu.com/article/3ea51489c045d852e61bbaab.html>

JVM 在Java应用程序优化中是不可缺少的一大重项，如何合理配置Java参数，如果验证配置参数的有效性，从GC日志中可以获得很重要的提示，以下是笔者对GC垃圾收集器默认开启的组合日志的部分的解析，希望能帮到想学习的同学O(∩\_∩)O~~

## 工具/原料

* jdk1.6
* -XX:+PrintGCDetails
* XX:+PrintGCTimeStamps
* -Xloggc:filename
* glassfish

## 方法/步骤

1. Hotspot JVM GC 垃圾收集器的默认默认开启的组合为：



1. 使用GC日志命令行选项为：

-XX:+PrintGCTimeStamps

-XX:+PrintGCDetails

-Xloggc:<filename>

1. **JVM 【-server】**

glassfish应用服务器  -server 启动 垃圾收集器默认组合方式为

新生代：Parallel Scavenge 并行回收GC

年老代和持久代：Parallel Old并行GC

1. **GC日志打印信息**：

-XX:+PrintGCTimeStamps输出格式：

289.556: [GC [PSYoungGen: 314113K->15937K(300928K)] 405513K->107901K(407680K), 0.0178568 secs] [Times: user=0.06 sys=0.00, real=0.01 secs]

293.271: [GC [PSYoungGen: 300865K->6577K(310720K)] 392829K->108873K(417472K), 0.0176464 secs] [Times: user=0.06 sys=0.00, real=0.01 secs]

详解：

293.271是从jvm启动直到垃圾收集发生所经历的时间，GC表示这是一次Minor GC（新生代垃圾收集）；[PSYoungGen: 300865K->6577K(310720K)] 提供了新生代空间的信息，PSYoungGen，表示新生代使用的是多线程垃圾收集器Parallel Scavenge。300865K表示垃圾收集之前新生代占用空间，6577K表示垃圾收集之后新生代的空间。新生代又细分为一个Eden区和两个Survivor区,Minor GC之后Eden区为空，6577K就是Survivor占用的空间。

括号里的310720K表示整个年轻代的大小。

392829K->108873K(417472K),表示垃圾收集之前（392829K）与之后（108873K）Java堆的大小（总堆417472K，堆大小包括新生代和年老代）

由新生代和Java堆占用大小可以算出年老代占用空间，如，Java堆大小417472K，新生代大小310720K那么年老代占用空间是417472K-310720K=106752k；垃圾收集之前老年代占用的空间为392829K-300865K=91964k 垃圾收集之后老年代占用空间108873K-6577K=102296k.

0.0176464 secs表示垃圾收集过程所消耗的时间。

 [Times: user=0.06 sys=0.00, real=0.01 secs] 提供cpu使用及时间消耗，user是用户模式垃圾收集消耗的cpu时间，实例中垃圾收集器消耗了0.06秒用户态cpu时间，sys是消耗系统态cpu时间,real是指垃圾收集器消耗的实际时间。

1. -XX:+PrintGCDetails输出格式：

293.289: [Full GC [PSYoungGen: 6577K->0K(310720K)]

[PSOldGen: 102295K->102198K(134208K)] 108873K->102198K(444928K)

[PSPermGen: 59082K->58479K(104192K)], 0.3332354 secs]

[Times: user=0.33 sys=0.00, real=0.33 secs]

说明：

Full GC表示执行全局垃圾回收

[PSYoungGen: 6577K->0K(310720K)] 提供新生代空间信息，解释同上

[PSOldGen: 102295K->102198K(134208K)]提供了年老代空间信息；

108873K->102198K(444928K)整个堆空间信息；

[PSPermGen: 59082K->58479K(104192K)]提供了持久代空间信息；

1. **JVM【-client】**

glassfish应用服务器 默认 -client启动 垃圾收集器默认组合方式为

新生代：Serial串行GC  (新生代使用-XX:+UseSerialGC收集器)

年老代和持久代：Serial Old 串行GC

1. GC日志打印信息：

-XX:+PrintGCTimeStamps输出格式：

9.007: [GC 9.007: [DefNew: 13526K->1060K(14400K), 0.0041350 secs] 44692K->32226K(46108K), 0.0041767 secs] [Times: user=0.01 sys=0.00, real=0.01 secs]

9.259: [GC 9.259: [DefNew: 13924K->1089K(14400K), 0.0042760 secs] 45090K->32650K(46108K), 0.0043143 secs] [Times: user=0.02 sys=0.00, real=0.02 secs]

说明：

DefNew表示新生代使用Serial串行GC垃圾收集器，defNew提供新生代空间信息；

DefNewGeneration 就是 default new generation

1. 8

-XX:+PrintGCDetails输出格式：

9.348: [GC 9.348: [DefNew: 13953K->976K(14400K), 0.0040943 secs]9.352: [Tenured: 32163K->32220K(32220K), 0.1182207 secs] 45514K->32338K(46620K), [Perm : 30467K->30467K(65536K)], 0.1224318 secs] [Times: user=0.12 sys=0.00, real=0.13 secs]

DefNew：表示新生代使用Serial串行GC垃圾收集器，defNew提供新生代空间信息；

Tenured：提供年老代空间信息；

45514K->32338K(46620K)：整个堆空间大小信息；

Perm ：提供持久代空间信息；