GC 日志格式和参数汇总

CMS日志分析

执行命令：java -Xms20M -Xmx20M -XX:+PrintGCDetails -XX:+UseConcMarkSweepGC com.mashibing.jvm.gc.T15_FullGC_Problem01

UseConcMarkSweepGC ：这个参数就可以指定CMS Old区

```
年轻代：
[GC (Allocation Failure) [ParNew: 6144K->640K(6144K), 0.0265885 secs] 6585K->2770K(19840K), 0.0268035 secs] [Times: user=0.02 sys=0.00, rea
/**
ParNew：年轻代收集器
6144->640：收集前后的对比
（6144）：整个年轻代容量6585 -> 2770：整个堆的情况
（19840）：整个堆大小
**/
```

```
Old
[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（最大）
    //9866 (19840) : 整个堆使用（最大）
[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]
    //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 unlo
    //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:
        //cleaning up symbol and string tables which hold class-level metadata and
        //internalized string respectively
    //CMS-remark: 8511K(13696K): 阶段过后的老年代占用及容量
    //10108K(19840K): 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，为下次GC做准备
```

G1 YGC + mixedGC + FullGC

YGC—— STW Y 区

mixedGC —— 达到设置比例后 Y + O区

FGC —— 单线程 Old G1的调优目标是FullGC

G1有一个设置暂停时间的参数，目标停顿时间（比如设定了20ms），这个时间的依据是YGC的回收时间，G1会朝着自己设置的这个时间去努力优化。Y区占的比例默认50% - 60% ，如果在回收过程中发现回收Y区的时间大概是50ms，这个时候G1会自动调整Y区占比比例，一直打到我们设置的停顿时间。自动分区调整

i. ▪https://www.oracle.com/technical-resources/articles/java/g1gc.html

G1日志详解

```
[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]
         [Processed Buffers:  1]
      [Scan RS (ms):  0.0]
      [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]
   [Eden: 0.0B(1024.0K)->0.0B(1024.0K) Survivors: 0.0B->0.0B Heap: 18.8M(20.0M)->18.8M(20.0M)]
 [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]

// G1 下 出现Full GC  就是很严重的问题
```

GC常用参数

- -Xmn -Xms -Xmx -Xss
  年轻代 最小堆 最大堆 栈空间
- -XX:+UseTLAB ———— 默认 不建议调
  使用TLAB，默认打开
- -XX:+PrintTLAB ———— 默认 不建议调
  打印TLAB的使用情况
- -XX:TLABSize ———— 默认 不建议调
  设置TLAB大小
- -XX:+DisableExplictGC ———— 默认 不建议调
  System.gc()不管用 ，配置了这个参数后，这条语句不起作用

  System.gc() 这条语句在jvm中表示FGC，建议JVM进行GC，调用后不立即GC
- -XX:+PrintGC 打印GC
- -XX:+PrintGCDetails 打印GC详细信息
- -XX:+PrintHeapAtGC GC的时候打印堆栈的详细情况
- -XX:+PrintGCTimeStamps 打印发生GC的时候系统的时间
- -XX:+PrintGCApplicationConcurrentTime (重要性低)
  打印应用程序时间

- -XX:+PrintGCApplicationStoppedTime （重要性低）
  打印暂停时长
- -XX:+PrintReferenceGC （重要性低）
  记录回收了多少种不同引用类型的引用
- -verbose:class
  类加载详细过程　　打印类加载的详细过程
- XX:+PrintVMOptions　　打印JVM运行时的参数
- XX:+PrintFlagsFinal

  必须会用　如 java -XX:PrintFlagsFinal -version | grep G1　查看G1 相关参数
- -XX:+PrintFlagsInitial　初始化默认的参数
  必须会用
- -Xloggc:opt/log/gc.log　　记录GC日志的
- -XX:MaxTenuringThreshold
  GC升代年龄，最大值15
- 锁自旋次数 -XX:PreBlockSpin 热点代码检测参数-XX:CompileThreshold 逃逸分析 标量替换 ...
  这些不建议设置

Parallel常用参数

- -XX:SurvivorRatio————— 默认比例Eden: S0: S1 = 8:1:1 可以自己调整， 一般很少调整
- -XX:PreTenureSizeThreshold
  大对象到底多大，有些大对象是直接分配到 Old区的，可以通过这个参数设置大对象标准
- -XX:MaxTenuringThreshold
- -XX:+ParallelGCThreads
  并行收集器的线程数，同样适用于CMS，一般设为和CPU核数相同　一般不调整
- -XX:+UseAdaptiveSizePolicy
  自动选择各区大小比例

CMS常用参数

- XX:+UseConcMarkSweepGC　——-CMS 启动参数
- XX:ParallelCMSThreads
  CMS线程数量
- XX:CMSInitiatingOccupancyFraction　———— 达到这个比例后会 CMS GC 会工作
  使用多少比例的老年代后开始CMS收集，默认是68%(近似值)，

  如果频繁发生SerialOld卡顿，应该调小，（频繁CMS回收），这个值设置的过大，预留空间变小，在工作现场分配了好多对象的时候，容易装不下，CMS回收不及时，会发生卡顿，GC线程SerilOld 工作，STW 停顿，工作现场等待
- XX:+UseCMSCompactAtFullCollection
  在FGC时进行压缩
- XX:CMSFullGCsBeforeCompaction
  多少次FGC之后进行压缩
- XX:+CMSClassUnloadingEnabled　回收方法去不用的Class
- XX:CMSInitiatingPermOccupancyFraction
  达到什么比例时进行Perm回收
- GCTimeRatio
  设置GC时间占用程序运行时间的百分比

- XX:MaxGCPauseMillis
  停顿时间，是一个建议时间，GC会尝试用各种手段达到这个时间，比如减小年轻代

G1常用参数

- XX:+UseG1GC　　启用G1
- XX:MaxGCPauseMillis　Y区停顿时间的期望值
  建议值，G1会尝试调整Young区的块数来达到这个值
- XX:GCPauseIntervalMillis
  ？GC的间隔时间
- XX:+G1HeapRegionSize
  分区大小，建议逐渐增大该值，1 2 4 8 16 32。
  随着size增加，垃圾的存活时间更长，GC间隔更长，但每次GC的时间也会更长

  块越小，每次GC间隔较小，但是GC频率会增大
  ZGC做了改进（动态区块大小）
- G1NewSizePercent　　G1 Y去大小是动态调整的，调整范围是 5% - 60%
  新生代最小比例，默认为5%
- G1MaxNewSizePercent
  新生代最大比例，默认为60%
- GCTimeRatio
  GC时间建议比例，G1会根据这个值调整堆空间
- ConcGCThreads
  线程数量
- InitiatingHeapOccupancyPercent
  启动G1的堆空间占用比例

参考资料

i. https://blogs.oracle.com/jonthecollector/our-collectors
ii. https://docs.oracle.com/javase/8/docs/technotes/tools/unix/java.html
iii. http://java.sun.com/javase/technologies/hotspot/vmoptions.jsp
iv. JVM调优参考文档：https://docs.oracle.com/en/java/javase/13/gctuning/introduction-garbage-collection-tuning.html#GUID-8A443184-7E07-4B71-9777-4F12947C8184
v. https://www.cnblogs.com/nxlhero/p/11660854.html 在线排查工具
vi. https://www.jianshu.com/p/507f7e0cc3a3 arthas常用命令
vii. Arthas手册：
    1. 启动arthas java -jar arthas-boot.jar
    2. 绑定java进程
    3. dashboard命令观察系统整体情况
    4. help 查看帮助
    5. help xx 查看具体命令帮助
viii. jmap命令参考： https://www.jianshu.com/p/507f7e0cc3a3

1. jmap -heap pid
2. jmap -histo pid
3. jmap -clstats pid