GC垃圾收集器&JVM调优汇总

Java极客技术 今天

编者荐语：

全网最全JVM调优汇总，推荐收藏～

以下文章来源于Java技术指北 ， 作者指北君

Java技术指北

回复：java，获取精华资料。专注分享Java技术干货、Java 技术、Spring 全家桶、Jav...

收藏最近朋友小B说想一份GC优化的资料内容，JVM内存分析以及性能调优的时候方便查询。可是文章太多根本看不过来，那么今天指北君就为大家带来这份，GC垃圾收集器总结及其优化指南，让你的JVM从此不再寂寞。

PS:最近又赶上跳槽的高峰期，好多粉丝，都问我要有没有最新面试题，我连日加班好多天，终于整理好了，16000+ 道，295多份，多份面试题大全，我会持续更新中，马上就会整理更多！【文末有领取方式】

1 垃圾收集器

1.1 Serial/Serial Old 收集器：

Serial 收集器作用于年轻代中， 采用 复制算法 ， 属于串行回收方式

Serial Old 收集器 采用串行回收， STW机制， 采用 标记-压缩 算法 ，

"-XX:+UseSerialGC" 手动指定Serial收集器执行内存回收任务

"-XX:+PrintGCDetails" 年轻代串行收集器的工作日志开关

1.2 ParNew 收集器

ParnNew 可以说是Serial收集器的多线程版本， ParNew 收集器在 年轻代 中同样才用的也是复制算法 和 STW 机制 并行回收机制。

ParnNew 收集器的优势体现在多CPU，多核心的环境中， 在某些注重低延迟的应用场景下 ParNew 和 CMS 收集器的组合模式， 在Server 模式下的内存回收效果很好。

使用 "-XX:+UserParNewGC" 手动指定使用ParNew收集器 "

-XX:+UserParallelGC" 表示 年轻代使用并行垃圾回收器， 老年代使用串行收集器

1.3 Parallel/Parallel Old 收集器

Parallel收集器是并行回收 采用复制算法 年轻代 STW ， 和ParNew不同Parrllel收集器可以控制程序的吞吐量大小， 被称为-吞吐量优先的垃圾收集器.

Parallel Old采用标记整理算法，用于老年代的垃圾回收。

常用参数:

"-XX:+GCTimeRatio:N" 设置执行内存回收的时间所占JVM运行总时间的比例， 1/(1+N) 默认 N为99

"-XX:+MaxGCPauseMills" 设置执行内存回收STW 的暂停时间阈值， 若指定该值，则尽可能地在设定的时间内完成内存回收。

"-XX:+UseAdaptiveSizePolicy" 选项用于设置GC的自动分代大小调节策略。

"-XX:UseParallelOldGC"可在年轻代和老年代都是用并行回收收集器， 此收集器重点关注吞吐量

"-XX:ParallelGCThreads" 可用于设置垃圾回收时的线程数量

Parallel Old 收集器采用了 标记-压缩算法， 用于老年代垃圾回收， 并行回收 STW

Parallel 和 Parallel Old 收集器的组合 在Server 模式下的内存回收性能较好。

1.4 CMS(Concurrent-Mark-Sweep) 收集器

基于低延迟的考虑 ， 是并行垃圾回收器， 而且是老年代垃圾收集， 低延迟， 采用标记清除算法。会有短暂的STW

基本步骤如下 ：

1. 初始标记(Initial Mark) ：STW 标记根对象直接关联、可达的对象
2. 并发标记(Concurrent Mark) ：将不可达对象 标记为垃圾对象
3. 再次标记(Remark) ：STW 确保垃圾对象被成功且正确得标记
4. 并发清除(Concurrent Sweep)：垃圾回收

常用参数:

"-XX:+UseCMS-CompactAtFullCollection" 用于指定在执行完FullGC 之后 是否对内存空间进行压缩整理，

"-XX:+CMSFullGCs-BeforeCompaction" 设定在执行多少次FullGC 之后对内存空间进行压缩整理

"-XX:+CMSInitiatingOccupanyFraction" 设置老年代中的内存使用率达到多少百分比的时候执行内存回收 JDK1.6之前默认值为68% JDK1.6 之后默认92% ，CMS垃圾收集器在回收过程中程依然可能会产生垃圾，所以需要设定一个阈值来进行垃圾回收，如果CMS回收失败，JVM则会启动老年代串行收集器进行垃圾回收，程序的STW时间会较长， 所以可以在内存增长缓慢的程序里面设置较大阈值，在内存增长快速的程序里面设置较小的阈值， 避免触发老年代串行收集器。

"-XX:UseConMarkSweepGC" 表示年轻代使用并行收集器，老年代使用CMS 年轻代 并行收集器工作时的线程数量可以使用 "-XX:ParallelGCThreads" 选项指定， 一般最好与CPU的数量相当.

1.5 G1(Garbage-First) 收集器

G1将整个堆划分为若干个大小相等的区间（1-32MB）,Region类型分为Unused Region、Eden Region 和 Survivor Region组成了年轻代空间，Old Region， Humongous Region(里面的对象超过每个Region的50%)，

每个Region都会有一个Region Set （RS），RS的数据结构是Hash表，里面的数据是CardTable （堆中没512Byte 映射在 card table 1 byte ） ， 简单说 RS 里面存的是Region种存活对象的指针。当Region中数据发生变化的时候 ， 首先反映到Card Table 中的一个或者

对个card上，RS通过扫面内部的Card Table 得知Region中内存使用情况和存活对象，在使用Region过程中，如果一个Region被填满了，分配内存的线程会重新选择一个新的Region，空闲Region被组织到一个基于链表的数据结构（LinkedList ）中，这样可以快速找到Region.

G1 GC 可以分为 Young GC 和Mixed GC

年轻代GC：当年轻代达到一定的阈值，就开始年轻代的并发收集。将Eden Region中存活对象copy转移到Survivor的Region中，同时释放Eden Region 。存活的时间够久就移到Old Reagion。

Mixed GC：当Old Region占比达到一定的比例（通过 -XX:InitantingHeapOccupancyPercent 设置，默认45%）之后，会触发并行标记，然后就会进行Mixed GC。Mixed GC 对一个叫做CSet的Region集合进行垃圾回收，其中包含了所有的年轻代Region和选取的一部分回收效益最好的Old Region。

并发标记的过程类似于CMS中的标记过程。

G1 可选优化参数

1. 年轻代优化

   -XX:G1NewSizePercent Java 堆初始化大小 ，默认是整个Java堆大小的5%，

   -XX:G1MaxNewPercent 最大占用对内存的百分比，默认是60%

   -XX:MaxGCPauseMills 每次目标停顿时间, 默认200ms

可以根据上述三个参数的调整来优化年轻代的垃圾回收

2. 并行标记阶段优化

   并行标记阶段， -XX:InitantingHeapOccupancyPercent 决定了什么时候初始化并行标记循环 默认值45%

   -XX:ConcGCThreads 并发GC数量， 是-XX:ParallelGCThreads 的1/4 ，可以通过修改两个值来改变并线程的数量。

3. 混合回收阶段优化

-XX:+PrintAdaptiveSizePoliy 开启后会输出完成的GC生态日志，

-XX:G1HeapWastePercent 表示最大可忍受的垃圾总量，默认是堆空间的5%，如果Mixed GC占用的时间过多，可以将此值调整大一点。

-XX:G1MixedGCCountTarget 默认为8，表示Mixed GC过程中入选的Old Region的最小阈值，用于控制入选的Old Region的数量

-XX:G1OldCSetRegionThresholdPercent 默认值100% ，表示加入到CSet中的Old Region的最大数量。

-XX:G1MixedGCLiveThresholdPercent 默认值 85% ，表示设置的每CSet中每个区间最多存活对象的百分比。

2 JVM的一些常用参数

1. -XX:+PrintGCDetails 用于记录GC运行时的详细信息并输出。
2. -verbose:gc -Xloggc:gc.log 结合前一个选项可以在项目根目录下生成gc.log文件记录gc详细信息
3. -XX:+UseSerialGC 使用串行gc收集器，使用与小于100MB的内存空间场景
4. -XX:+UserParNewGC 独占式GC ， 多线程GC （未来不可用）
5. -XX:+UseParallelGC
6. -XX:+UseParallelOldGC
7. -XX:+UseConcMarkSweepGC
8. -XX:+UseG1GC
9. -XX:+PrintGCApplicationStoppedTime 输出GC造成应用程序暂停的时间
10. -XX:+PrintGCApplicationConcurrentTime 与上面参数结合使用
11. -XX:+ConcGCThreads=4 设置Java应用程序线程并行执行的GC线程数量 若设置的值超过JVM允许GC并行线程的数量则报错， 默认的并行标记线程数量计算如下 ConGCThreads = Max((ParallelGCThreads#+2)/4,1)
12. -XX:G1HeapRegionSize G1 GC 独有，Region大小默认为堆的1/2000 也可以设置 1MB 2MB 4MB 8MB 16MB 32MB 等6个档次
13. -XX:G1HeapWastePercent=5 控制G1GC 不会回收的空闲内存比例，默认是堆内存的5% G1 GC在回收过程中会回收所有Region的内存，并持续地进行回收工作，直到内存空闲

比例达到次值

14. -XX:G1MixedGCCountTarget=8 老年代Region 的回收时间通常来说比年轻带Region回收时间长，此选项设置并行循环之后启动多少个混合GC 默认值是 8个 设置一个比较大的值可以让G1 GC在老年代Region回收时多花一些时间，如果一个混合GC停顿的时间很长，说明它要做的使很多，所以可以增大这个值的设置，但这个值过大的话，会造成并行循环等待混合GC完成的时间也相应的增加。
15. -XX:G1PrintRegionLivenessInfo 开启这个选项会在标记循环之后输出详细信息（诊断选项） 在使用之前需要开启-XX:UnlockDiagnosticVMOptions 选项， 此选项会打印内存内部每个Region里面存活的对象信息， 包括使用率 RSET大小、回收一个Region的价值（性价比） 15. -XX:G1ReservePercent=10 此选项默认保留对内存的10%，用于某个对象进入下一个阶段，预留内存空间不可用于年轻带
16. -XX:+G1SummarizeRSetStats 打印每个Region的详细信息。 此选项和-XX:G1PrintRegionLivenessInfo选项一样，是一个诊断选项也需要开启 -XX:UnlockDiagnosticVMOptions 选项 。
17. -XX:+G1TraceConcRefinement 诊断选项，启动这个选项，并行Refinement线程相关的信息会被打印，线程启动和结束时，信息都会被打印。
18. -XX:G1UseAdaptiveConcRefinement 默认开启的选项， 它会动态地对每一次GC中-XX:G1ConcRefinementGreenZone、-XX:G1ConcRfinementYellowZone、-XX:-XX:G1ConcRfinementRedZone，的值进行重新计算 并行Refinement线程是持续运行的，并且会随着update log buffer 积累的数量而动态调节， -XX:G1ConcRefinementGreenZone、-XX:G1ConcRfinementYellowZone、-XX:-XX:G1ConcRfinementRedZone，三个选项是用来根据不同的buffer使用不同的Refinement线程，其作用就是保证Refinement线程尽可能更上update log buffer生产的的步伐
19. -XX:GCTimeRatio=9 这个选项代表Java应用程序话费时间与GC线程花费时间的比率，1/(1+GCTimeRatio) 默认值是9 表示花费在GC工作量上的时间占总时间的10%
20. -XX:+HeapDumpBeforeFullGC/-XX:+HeapDumpAfterFullGC 启用此选项， 在Full GC开始之前有一个hprof文件会被创建， 两个选项同时使用，可以对比Full GC前后的java堆内存，找出内存泄漏以及其他问题，
21. -XX:InitiatingHeapOccypancyPercent=45 该选项默认值是45 表示G1 GC并行循环初始设置的堆大小值， 这个值决定了一个并行循环是不是要开始执行， 它的逻辑是在一次GC完成后，标胶老年代占用的空间和整个Java堆之间的比例，如果大于这个值，则预约下一次GC开始一个并行循环回收垃圾，从初始标记阶段开始。 这个值越小,GC 越频繁，反之 值越大，
22. -XX:UseStringDeduplication 该选项启动String对象的去重工作，默认不启用。 如果启用该选项 String1.equals（String2） 如果两个对象包含相同的内容则返回true
23. -XX:StringDeduplicationAgeThreshold=3 针对-XX:UseStringDeduplication选项，默认值为3 字符串对象的年龄超过设定的阈值，或者提升到G1 GC老年代Region之后，就会

成为字符串去重的候选对象，去重操作只会有一次。

24. -XX:PrintStringDeduplicationStatistics 可以通过读取输出的统计资料来了解是否字符串去重后节约了大量的堆内存空间，默认关闭
25. -XX:+G1UseAdaptiveHOP JDK9的新选项 默认启用，通过动态调节标记阶段开始的时间，以达到提升应用程序吞吐量的目标， 主要通过尽可能迟地触发标记循环方式来避免消耗老年代空间，
26. -XX:MaxGCPauseMills=200 【重要选项】设置G1的目标停顿时间 ，单位为ms 默认值为200ms
27. -XX:MinHeapFreeRatio=40 设置对内可以空闲的最小的内存空间大小，默认为堆内存的40% 当空闲堆内存大小小于此值的时候，需要判断-Xms 和 -Xmx 两个初始化设置值，如果-Xms 和 -Xmx 不一样，那么就有机会扩展堆内存，否则就无法扩展。
28. -X:MaxHeapFreeRatio =70 这只最大空闲空间大小，默认为堆内存的70%，当大于这个空闲比率的时候 G1 GC 会自动减少对内存的大小， 需要判断-Xms 和 -Xmx的大小 如果两者一样则有机会减小堆内存，否则无法减小堆内存
29. -XX：+PrintAdaptiveSizePolicy 这个选项决定是否开启堆内存大小变化的相应记录信息打印， 是否打印这些星系到GC日志里面
30. -XX:+ResizePLAB GC使用的本地线程分配缓存块采用动态值还是静态值 默认开启 31. -XX:+ResizeTLAB Java线程使用的本地线程分配缓存块采用 动态值还是静态值。默认开启
31. -XX:+ClassUnloadingWithCncurrent 开启G1 GC在并行循环卸载类，尤其是在老年代的并行回收阶段，默认是开启的。 这个选项开启后会在并行循环的重标记阶段卸载JVM没有用到的类， 这些工作也可以放到Full GC 里面去做。是否开启此选项要看性价比。如果GC停顿时间比我们设置的最大GC停顿目标时间还长，并且需要卸载的类也不多，建议关闭此选项
32. -XX:+ClassUnloading 默认是是True 决定JVM是否会卸载无用的类，如果关闭此选项，无论是并行回收循环还是Full GC 都不会再卸载这些类，所以需要谨慎关闭
33. -XX:+UnloadingDiagnosticVMOptions 是否开启诊断选项，默认值是 False
34. -XX:+UnlockExperimentalVMOptions 默认关闭
35. -XX:+UnlockCommercialFeatures 是都使用Oracle特有特性， 默认关闭

3 JVM常用调优分析工具

JVM内存分析调优，如果能多多利用调优的一些工具，那将会事半功倍。那么指北君问大家介绍比较好用的一些调优工具，包括Java自带的和第三方的一些好用工具。

jps

JVM Process Status Tool，查询当前运行状态的Java 进程

语法格式

```
jps [<options>]
```

具体 [options]选项：-q：仅输出VM标识列表；-m：输出main方法的参数，输出可能为null；-l：输出完整的包名，应用main calss，jar的完全路径名；-v：输出JVM参数 ；-V：输出通过flag文件传递到JVM中的参数(.hotspotrc文件或-XX:Flags=所指定的文件 ；

jinfo

JVM Configuration info ， 查询Java进程运行时的详细信息以及JVM的参数详细信息

命令格式

```
jinfo [ option ] pid
jinfo [ option ] executable core
jinfo [ option ] [server-id@]remote-hostname-or-IP
```

具体 [options]选项：

-flag *name* ： prints the name and value of the given command line flag.

-flag [+|-]name ： enables or disables the given boolean command line flag.

-flag name=value ： sets the given command line flag to the specified value.

-flags prints ： command line flags passed to the JVM. pairs.

-sysprops ： prints Java System properties as name, value pairs.

示例:

```
bin>jinfo -flags 17876
```

```
VM Flags:
-XX:CICompilerCount=2 -XX:ConcGCThreads=1 -XX:ErrorFile=C:\Users\Michael\java_error_in_idea
```

```
bin>jinfo -flag InitialHeapSize 17876
-XX:InitialHeapSize=134217728
```

jstat

JVM Statistics Monitoring Tool 可查看JVM的性能统计信息，能显示出虚拟机进程中的类装载、内存、垃圾收集、JIT编译等运行数据。可参考官网详情：https://docs.oracle.com/javase/7/docs/technotes/tools/share/jstat.html

命令格式

```
jstat [ generalOption | outputOptions vmid [interval[s|ms] [count]] ]
```

generalOption 通用命令行参数，

outputOptions 一个过多个输出选项，

vmid 虚拟机定位标识，确定虚拟机运行的机器端口等

interval 查询并显示信息的时间间隔

count 需要查询信息的数量

关于jstat的参数选项有如：

Option	Displays...
class	Statistics on the behavior of the class loader.
compiler	Statistics of the behavior of the HotSpot Just-in-Time compiler.
gc	Statistics of the behavior of the garbage collected heap.
gccapacity	Statistics of the capacities of the generations and their corresponding spaces.

Option	Displays...
gccause	Summary of garbage collection statistics (same as **-gcutil**), with the cause of the last and current (if applicable) garbage collection events.
gcnew	Statistics of the behavior of the new generation.
gcnewcapacity	Statistics of the sizes of the new generations and its corresponding spaces.
gcold	Statistics of the behavior of the old and permanent generations.
gcoldcapacity	Statistics of the sizes of the old generation.
gcpermcapacity	Statistics of the sizes of the permanent generation.
gcutil	Summary of garbage collection statistics.
printcompilation	HotSpot compilation method statistics.

示例：

```
bin>jstat -gcoldcapacity -t 17876 250 3
Timestamp          OGCMN        OGCMX         OGC           OC        YGC    FGC    FGCT
         6378.7          0.0    1042432.0     505856.0     505856.0    310     0     0.000
         6379.0          0.0    1042432.0     505856.0     505856.0    310     0     0.000
         6379.3          0.0    1042432.0     505856.0     505856.0    310     0     0.000
```

jmap

JVM Memory Map 输出对象内存信息，Java堆信息等

命令格式

```
jmap [ option ] pid
jmap [ option ] executable core
jmap [ option ] [server-id@]remote-hostname-or-IP
```

options选项

-dump:[live,]format=b,file= 将Java堆生成dump文件，

-finalizerinfo 输出等待执行finalizer方法的对象

-heap 输出堆的详细信息，GC算法，堆参数等等

-histo[:live] 输出堆的统计信息，包括Java类，对象数量，内存size等

-permstat Prints class loader wise statistics of permanent generation of Java heap

-F 强制生成dump，与jmap -dump / jmap -histo 等一起使用，如果进程id没有响应则强制生成dump文件

示例：

```
bin>jmap -finalizerinfo 17876
No instances waiting for finalization found
```

生成Java 堆dump文件

```
bin>jmap -dump:format=b,file=E:/Java/heap/17876.hprof 17876
Heap dump file created
```

jstack

主要输出 Java线程的堆栈信息

```
jstack [options] pid
```

示例：

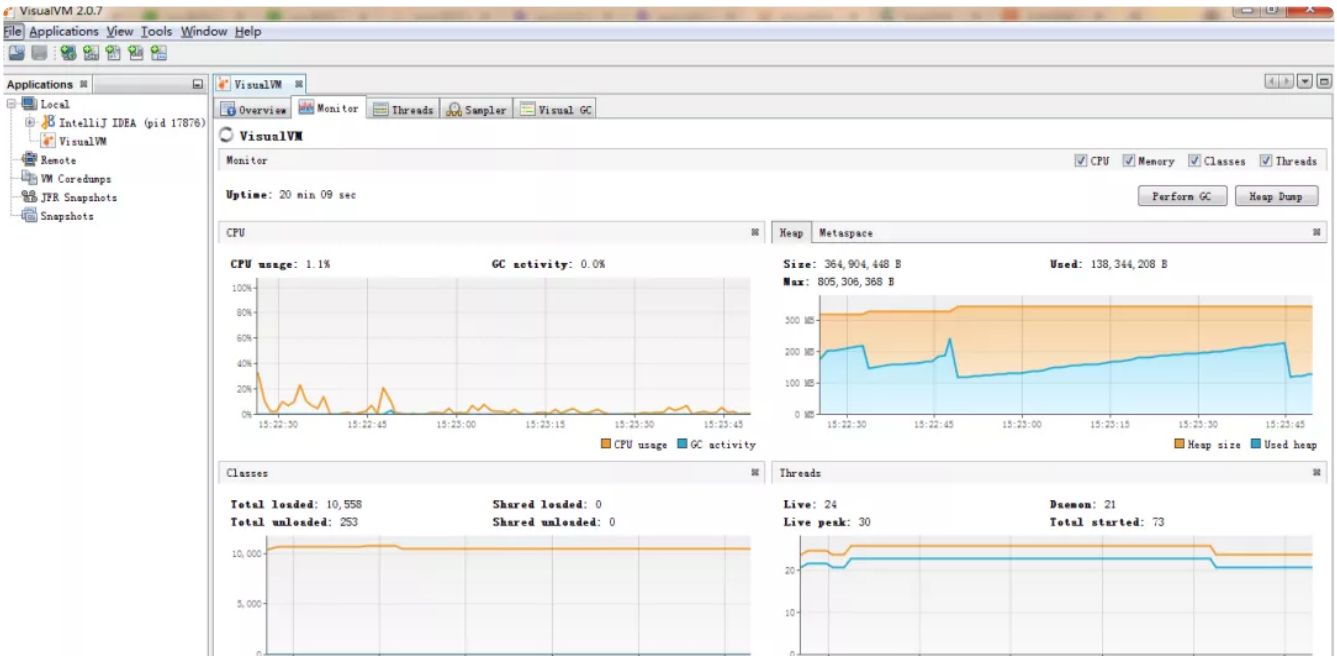
```
bin>jstack -l 17876
```

JConsole

JConsole 是Java自带的可视化程序，可以监控一些内存线程等信息，使用也比较方便。

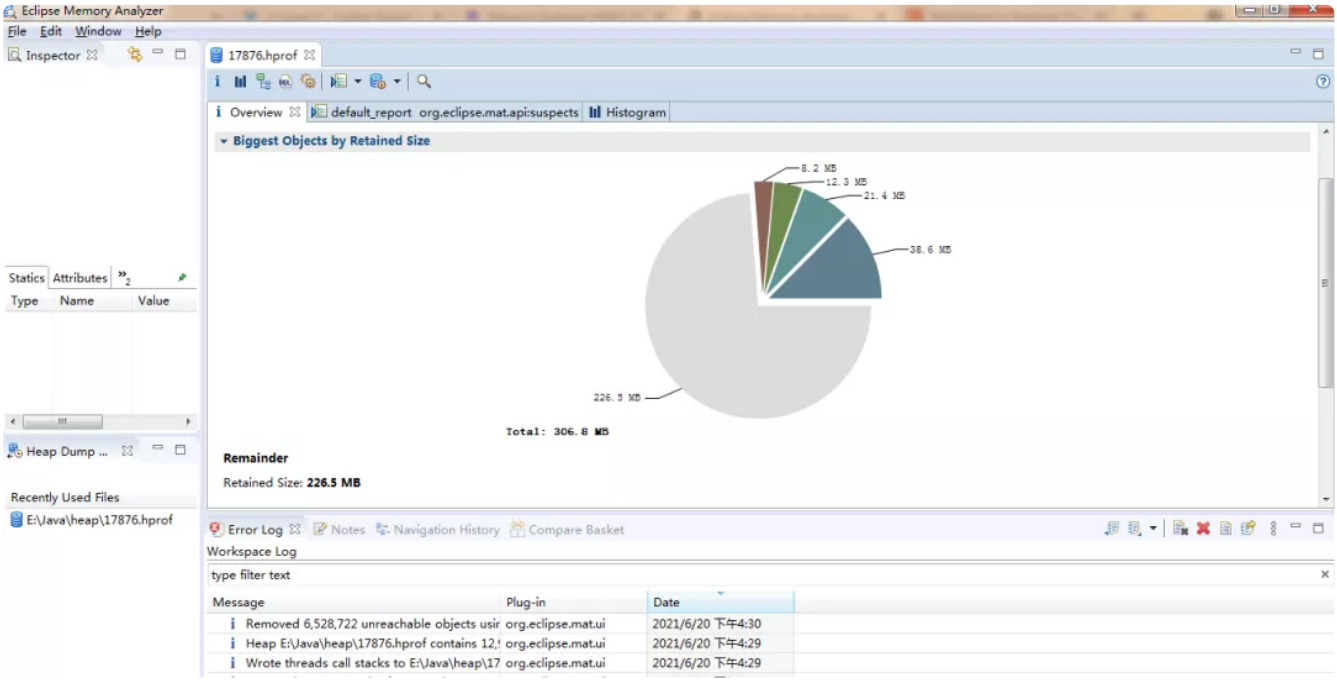
VisualVM

VisualVM在在Java8之后就被从JDK中拿走，如果想使用可以去VisualVM官网（文末贴出下载地址）下载，它有标准版和基于GraalVM的版本，同时还有IDEA的插件用起来也比较方便。我们下载标准版VisualVM2.07先看一下。还是熟悉的味道。



Memory Analyzer (MAT)

Eclipse 出品的一款Java堆内存分析程序，可以帮助我们找到内存泄漏和减少内存消耗。可在其官网下载，文末贴出下载地址



总结

本篇总结了一些常用GC的垃圾回收流程，GC调优可以参考的一些参数。另外还介绍了一些分析Java性能的一下Java命令及可视化分析工具，在实际生产环境使用的话肯定会比所介绍的还要复杂，所以看官们还不快快动起手来操练一下。让你的JVM不再寂寞。由于每一个工具的详细使用都需要大篇幅，那么指北君后面将会为大家一一带来这些工具如何进行JVM性能调优。敬请期待！

后记

VisualVM 官网：https://visualvm.github.io/

GraalVM 官网 ： https://www.graalvm.org/

Memory Analyzer (MAT) 下载 官网：https://www.eclipse.org/mat/downloads.php （选择国内中科大的镜像，下载会快许多）

面试大全包括：包括 Java 集合、JVM、多线程、并发编程、设计模式、SpringBoot、SpringCloud、Java、MyBatis、ZooKeeper、Dubbo、Elasticsearch、Memcached、MongoDB、Redis、MySQL、RabbitMQ、Kafka、Linux、Netty、Tomcat、Python、HTML、CSS、Vue、React、JavaScript、Android 大数据、阿里巴巴等大厂面试题等、等技术栈!

领取方式：扫描下方公众号【Java技术指北】回复【面试题】即可获取