Java内存查看与分析

作者：Eddie
原文链接：https://ld246.com/article/1469065540696
来源网站：链滴

Java运行时数据区包含：Method area， heap，java stacks， pc register，native method stacks
有些内存块在程序所有内存中共享 （Method area， heap），些只对单个线程有效 （java stacks， pc register，native method stacks）。

业界有很多强大的java profile的工具，比如Jporfiler，yourkit，这些收费的东我就不想说了，想说的是，其实java自己就提供了很多内存监控的小工具，下面列举的工具只是一小分，仔细研究下jdk的工具，还是蛮有意思的呢：)
1．gc日志输出
在 jvm启动参数中加入 -XX:+rintGC -XX:+PrintGCDetails -XX:+PrintGCTimestamps -XX:+PrintGCApplicationStopedTime，jm将会按照这些参数顺序输出gc概要信息，详细信息，gc时间信息，gc造成 的应用暂停时间。如果刚才的参数后面加入参数 -Xloggc:文件路径，gc信息将会输出到指定的文件中。其他参数还有-verboe:gc和 -XX:+PrintTenuringDistribution等。

2：jconsole
jconsole是jdk自带的一个内存分析工具，它提供了图形界面。可以查看到被监控的jvm内存信息，线程信息，类加载信息，MBean信息。
jconsole 位于jdk目录下bin目录，在windows下是jconsole.exe，在unix和linux下是jconsole.sh，jconsole可 以监控本地应，也可以监控远程应用。 要监控本地应用，执行jconsole pid，pid就是运行的java进程id，如果不上pid参数，则执行jconsole命令后，会看到一个对话框弹出，上面列出了本地的java进 程，可以选一个进行监控。如果要远程监控，则要在远程服务器的jvm参数里加入一些东西，因为jconsole的远监控基于jmx的，关于 jconsole详细用法，请见专门介绍jconsle的文章，我也会在博客里专门详细介jconsole。

3:jviusalvm
在JDK6 update 7后，jdk推出了另外一个工具：jvisualvm,java可视化虚拟机，它不但提供了jconsole类似的功能，还供了jvm内存和cpu实 时诊断，还有手动dump出jvm内存情况，手动执行gc。 和jconsole一样，运行viusalvm，在jdk的bin目录下执行jviusalvm，windows下是 jviusalvm.exe,linux和unix下是jviusalv.sh。

4：jmap
jmap是jdk自带的jvm内存析的工具，位于jdk的bin目录。jdk1.6中jmap命令用法：
Usage:
jmap -histo(to connect to running process and print histogram of java object hea
jmap -dump:(to connect to running process and dump java heap)</spn>
dump-options:format=b binary default
file= dump hap to
Example: jmap -dump:format=b,file=heap.bin
<spn>jmap -histo 在屏幕上显示出指定pid的jvm内存状况。以我本机为例，执行该命令，屏幕显示：</pan>
num #instances #bytes class name

1: 24206 2791864
2: 2371 2145216 [C
3: 24206 1940648
4: 1951 136496
5: 26543 1282560
6: 6377 1081744 [B
7: 1793 909688
8: 1471 614624
9: 14581 548336 [Ljava.lang.Object;
10: 3863 513640 [I
<san>11: 20677 496248 java.lang.String
12: 3621 312776 [Ljava.util.HasMap$Entry;
13: 3335 266800 java.lang.reflect.Method
<pan>14: 8256 264192 java.io.ObjectStreamClass$WeakClassKey
15: 706 226112 java.util.TreeMap$Entry
16: 2355 173304 [S
<san>17: 1687 161952 java.lang.Class
18: 2769 150112 [[I
19: 3563 142520 java.util.HashMap
20: 5562 133488 java.util.HasMap$Entry
Total 239019 17140408
为了方便查，我删掉了一些行。从上面的信息很容易看出，#instance指的是对象数量，#bytes指的是这些对象用的内存大小，class name指的是对象类型。
再看jmap的dump选项，这选项是将jvm的堆中内存信息输出到一个文件中，在我本机执行
jmap -dump file=c:\dump.txt 340
注 意340是我本机的java进程pid，dump出来的文件较大有10几M，而且我只是开了tomcat，跑了一个很简单的应用，且没有任何访问，可以 想象，大繁忙的服务器上，dump出来的文件该有多大。需要知道的是，dump出来的文件信息是很原始的，不适合人直接观看，而jmap -histo显示的内容又太简单，例如只显示某些类型的对象占用多大内存以及这些对象的数量，但是没有更详细的信息，例如这些对象分别是由谁创建的。那 这么说，dump来的文件有什么用呢？当然有用，因为有专门分析jvm的内存dump文件的工具。

5：jhat
上 面说了，有很多工具都能分析jvm的内存dump文件，jat就是sun jdk6及以上版本自带的工具，位于jdk的bin目录，执行 jhat -J -Xmx512m [file] ，file就dump文件路径。jhat内置一个简单的web服务器，此命令执行后，jhat在命令行里显示分析结果的

问地址，可以用-port选项 指定端口，具体用法可以执行jhat -heap查看帮助信息。访问指定地址后就能看到页面上显示的信息，比jmap -histo命令显示的丰富得多，更为详细。

6：eclipse内存分析器
上 面说了jhat，它能分析jvm的dump文 件但是全部是文字显示，eclipse memory analyzer，是一个eclipse提供用于分析jvm 堆dump的插件网址为 http://www.eclipse.org/mat ,它的分析速度比jhat快，分析结果是图形界面显示，比jhat的读性更高。其实jvisualvm也可以分析dump文件，也是有图形界面显示的。

<pan>7：jstat
如 果说jmap倾向于分析jvm内存中对象信息的话，那么jsta是倾向于分析jvm内存的gc情况。都是jvm内存分析工具，但显然，它们是从不同维度 来分析的。jsa常用的参数有很多，如 -gc,-gcutil,-gccause，这些选项具体作用可查看jsat帮助信息，我经常用-gcutl，这个参数的作用不断的显示当前指定的 jvm内存的垃圾收集的信息。
我本机执行 jstat -gcutil 340 10000，这个命令是每个10秒钟输出一次jvm的gc信息，10000指的是间时间为10000毫秒。屏幕上显示如下信息(我只取了第一行，因为是按的一定频率显示，所以实际执行时候，会有很多行)：
S0 S1 E O P YGC YGCT FGC FGCT GCT
54.62 0.00 42.87 43.52 86.24 1792 5.093 33 7.670 12.763
额&ellip;… 怎么说呢，要看懂这些信息代表什么意思，还必须对jvm的gc机制有一定的了解才行啊其实如果对sun的 hot spot jvm的gc比较了解的人，应该很容易看懂这些信息，但是不清楚gc机制的，有点莫名其妙，所以在这里我还是先讲讲sun的jvm的gc机制吧。说到 gc，其实不仅仅只是java的念，其实在java之前，就有很多语言有gc的概念了，gc嘛就是垃圾收集的意思，更多的是一种算法性东西，而跟具体 语言没太大关系，所以关于gc的历史，gc的主流算法我就不讲了，那扯得太远了，得太远了就是扯淡。sun现在的jvm，内存的管理模型是分代模型，所 以gc当然是分代收集了。分代什么意思呢？就是将对象按照生命周期分成三个层次，分别是：新生代，旧生代，持久代。对象刚开分配的时候，大部分都在新 生代，当新生代gc提交被触发后了，执行一次新生代范围内的gc，这叫minor gc，如果执行了几次minor gc后，还有对象存活，将这些对象转入旧生代，因为这些对象已经经了组织的重重考验了哇。旧生代的gc频率会更低一些，如果旧生代执行了gc，那就是 full gc，因为是局部gc，而是全内存范围的gc，这会造成应用停顿，因为全内存收集，必须封锁内存，不许有新的象分配到内存，持久代就是一些jvm期 间，基本不会消失的对象，例如class的定义，jvm方法区信息例如静态块。需要主要的是，新生代里又分了三个空 间：eden，susvivor0，susvivor1，按字面上理解，就是伊甸园区，幸存1区，幸存2区。新对象分配在eden区中，eden区满 时，采用标记-复制法，即检查出eden区存活 的对象，并将这些对象复制到是s0或s1中，然后清空eden区。jvm的gc说来，不只是这么简单，例如还有串行收集，并行收集，并发收集，还有着名 的火车算法，不过那说得远了，现在对这个有大致了解就好。说到这里，再来看一下上面输出的信息：
S0 S1 E O P YGC YGCT FGC FGCT GCT
54.62 0.00 42.87 43.52 86.24 179 5.093 33 7.670 12.763
S0:新生代的susvivor0区，空间使用率为5462%</san>
S1:新生代的susvivor1区，空间使用率为0.00%(因为还没有执行第二次minor收集
E:eden区，空间使用率42.87%
O:旧生代，空间使率43.52%
P:持久带，空间使用率86.24%
YGC:mino gc执行次数1792次
YGCT:minor gc耗费的时间5.093毫秒
<pan>FGC:full gc执行次数33
FGCT:full gc耗费的时间7.670毫秒<b />GCT:gc耗费的总时间12.763毫秒
8. 怎样选择工具
上 面列举的一些工具，各有弊，其实如果在开发环境，使用什么样的工具是无所谓的，只要能得到结果就好。但是在生产环境里却不能乱选择，因为这些工具本身 就会耗费大量的系统资源，如果在一个生产服务器压力很大的时候贸然执行这些工具，可能会造成很意外的情况。最好不要在服务器本机监控，远程监控会比较好 一些但是如果要远程监控，服务器端的启动脚本要加入一些jvm参数，例如用jconsloe远程监控tomcat或joss等，都需要设置jvm的 jmx参数，如果仅仅只是分析服务器的内存分配和gc信息，强烈推荐，先用map导出服务器端的jvm的堆dump文件，然后再用jhat，或者 jvisualvm，或者eclipse内存分析器分析内存状况。
 转载http://tech.ifeng.com/internet/detail_2011_03/08/5022006_0.html