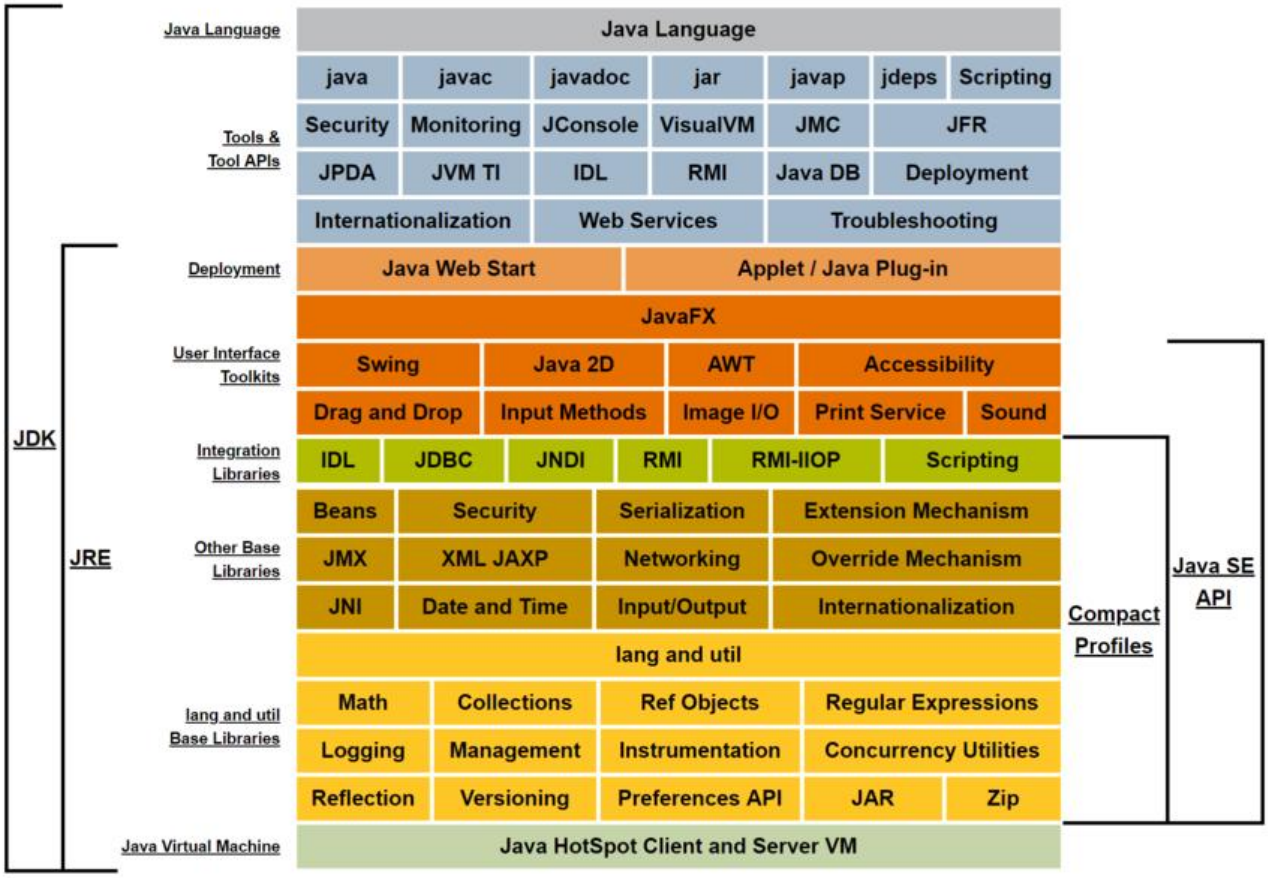
JDK 1.8 Hotspot 虚拟机参数说明

作者：xiaoweizha
原文链接：https://ld246.com/article/1552491323380
来源网站：链滴

有图为证

来自官网的Java Platform Standard Edition (Java SE) 8的概念图如下



官方文档https://docs.oracle.com/javase/8/docs/technotes/tools/unix/java.html#BABFAFAE

总的来说 可以配置的参数分为如下几类

具体配置，可以对比使用，个人推荐以官方文档为主，实践为辅

以下附常用的虚拟机参数配置

-Xmx

JVM最大允许分配的堆内存，按需分配

-Xms

JVM初始分配的堆内存，一般和Xmx配置成一样以避免每次gc后JVM重新分配内存。

-Xmn

年轻代内存大小，整个JVM内存=年轻代 + 年老代 + 元空间

-Xss

设置每个线程的堆栈大小

-XX:CMSClassUnloadingEnabled

HotspotJVM默认的gc是并行收集器，CMS相对于并行收集器不同的是CMS只回收老年代与永久代且CMS并不默认回收永久代需要开启 CMSClassUnloadingEnabled 参数。

元数据区与永久代的区别CMS垃圾回收器回收过程详解

ps: 永久代在jdk1.8之后被成为元数据区。

-XX:+UseCMSCompactAtFullCollection

原文链接：JDK 1.8 Hotspot 虚拟机参数说明

在Full GC时,开启对年老代的压缩.

-XX:CMSFullGCsBeforeCompaction=9

设置CMS GC在n次Full GC后进行内存压缩

-XX:CMSInitiatingOccupancyFraction=80

配置内存占比超过80%进行一次gc

-XX:+CMSParallelRemarkEnabled

降低标记停顿

gc停顿的意思就像是在整个分析期间冻结在某个时间点上，具体的原因是防止在分析的时候，对象引关系还在不断的变化，如果没有GC停顿很有可能分析不准确。

如何降低：在Serial的老年代垃圾收集器中，会把所有线程的暂停，停下来收集哪些是死亡对象。在CS和G1中都采取了初始标记、并发标记、短暂GC停顿重新标记，初始标记会直接记录能GC ROOTS联的对象，在并发标记的时候有一个线程来标记，这个时候对象的发生的变化都会记录下来，在重新记的时候会修正，这样就会降低GC停顿时间

-XX:+CMSScavengeBeforeRemark

CMS GC分为初始标记->并发标记->并发预清理->重新标记->并发清除->并发重置几个步骤,该参数作用是在Remark前对年轻代进行一次minor gc，以减轻Remark的工作量click me

-XX:+ExplicitGCInvokesConcurrent

打开此参数后，在做System.gc()时会做background模式CMS GC，即并行FULL GC，可提高FULL G效率

CMS并行fullGC

-XX:-HeapDumpOnOutOfMemoryError

可以让JVM在出现内存溢出时候Dump出当前的内存转储快照

-XX:HeapDumpPath=/data/applogs

DUMP文件的路径

-XX:InitialHeapSize=5361369088

初始堆内存大小

-XX:MaxNewSize=2061500416

最大新生代大小

-XX:MaxTenuringThreshold=9

在新生代中对象存活次数(经过Minor GC的次数)后仍然存活，就会晋升到老年代

-XX:NewSize=2061500416

新生代初始大小

-XX:OldPLABSize=16

老年代空间PLAB大小

对于OldPLAB的解释请参考 Old space PLABs

-XX:+PrintGC

输出GC日志

-XX:+PrintGCApplicationConcurrentTime

打印每次垃圾回收前，程序未中断的执行时间

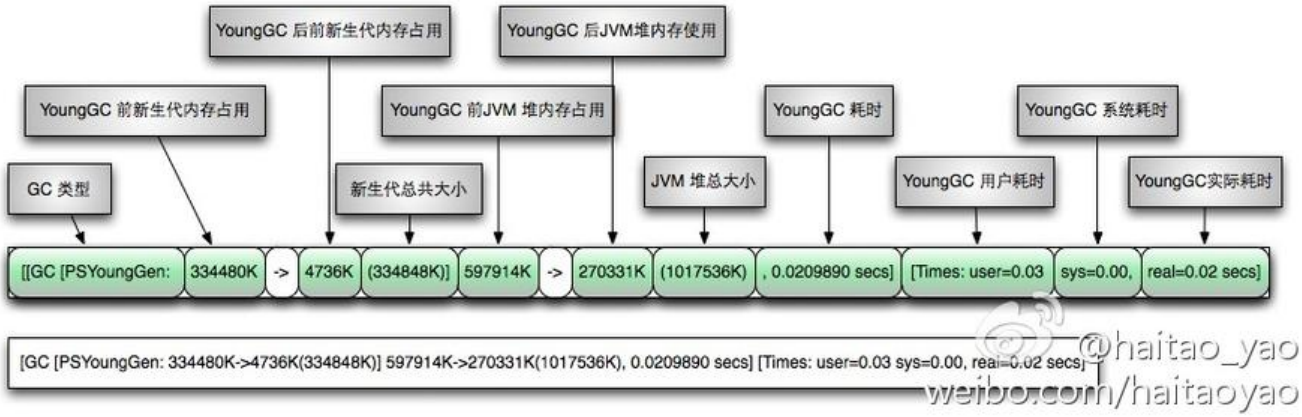
-XX:+PrintGCApplicationStoppedTime

输出GC应用暂停的时间

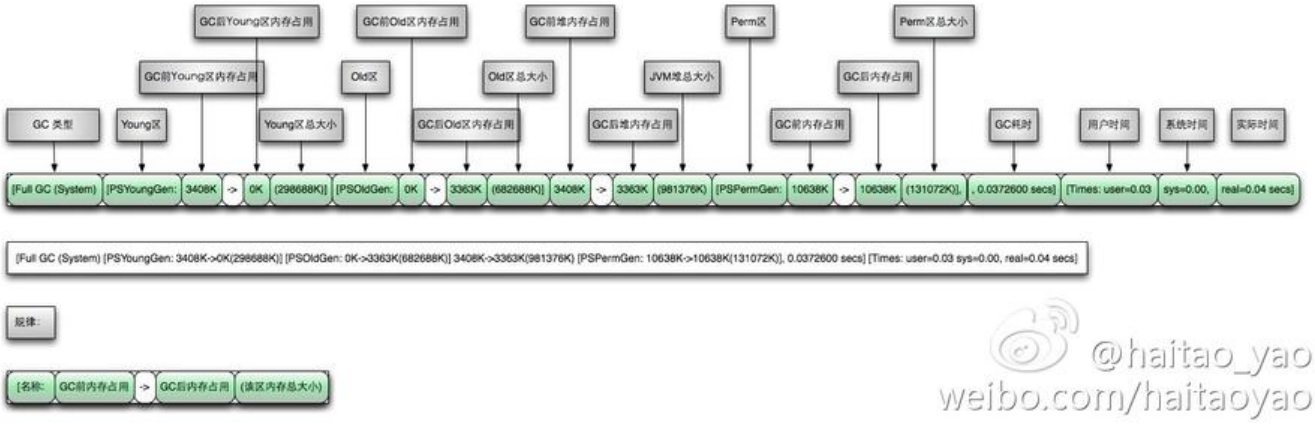
-XX:+PrintGCDetails

输出详细的GC日志

日志详细信息说明如下图示



原文链接：JDK 1.8 Hotspot 虚拟机参数说明



-XX:+PrintGCTimeStamps

输出gc时间戳

-XX:+PrintGCDetails

输出GC日期格式的时间戳

-XX:+PrintHeapAtGC

HotSpot在GC前后都会将GC堆的概要状况输出

-XX:-ReduceInitialCardMarks

解决gc bug

card-marking performance optimization算法在实现的时候有瑕疵，在某些情况下会引起heap corruption。这个情况主要发生在新创建的大对象

和Eden space大小差不多，然后jvm做young GC的时候。

解决方案：在启动参数中增加-XX:-ReduceInitialCardMarks将性能优化策略关闭。

-XX:+ScavengeBeforeFullGC

在Full gc前进行一次minor dc

-XX:SoftRefLRUPolicyMSPerMB=0

官方解释是：Soft reference在虚拟机中比在客户集中存活的更长一些。其清除频率可以用命令行参数-XX:SoftRefLRUPolicyMSPerMB=来控制，这可以指定每兆堆空闲空间的 soft reference 保持存活一旦它不强可达了）的毫秒数，这意味着每兆堆中的空闲空间中的 soft reference 会（在最后一个强用被回收之后）存活1秒钟。注意，这是一个近似的值，因为 soft reference 只会在垃圾回收时才会清除，而垃圾回收并不总在发生。系统默认为一秒，我觉得没必要等1秒，客户集中不用就立刻清除改为 -XX:SoftRefLRUPolicyMSPerMB=0;

-XX:SurvivorRatio=8

SurvivorRation 与 Eden区的比例 8：1

-XX:NewRatio=2

默认2 表示新生代占老年代的1/2，占整个堆的1/3；

-XX:MaxMetaspaceSize=

设置元空间允许大小，默认不受限制，JVM的Metaspace会进行动态扩展；

-XX:ThreadStackSize=512

线程堆栈大小

-XX:+UseCMSInitiatingOccupancyOnly

只是用设定的回收阈值(上面指定的70%),如果不指定,JVM仅在第一次使用设定值,后续则自动调整

用-XX+UseCMSInitiatingOccupancyOnly标志来命令JVM不基于运行时收集的数据来启动CMS垃圾收集周期。而是，当该标志被开启时，JVM通过CMSInitiatingOccupancyFraction的值进行每一次CS收集，而不仅仅是第一次。然而，请记住大多数情况下，JVM比我们自己能作出更好的垃圾收集决。因此，只有当我们充足的理由(比如测试)并且对应用程序产生的对象的生命周期有深刻的认知时，应该使用该标志。

-XX:+UseCompressedOops

由于64位JVM消耗的内存会比32位的大1.5倍，因为对象指针在64位架构下，长度会翻倍（更宽的寻）。好在从JDK 1.6 update14开始，64 bit JVM正式支持了 -XX:+UseCompressedOops 这个可以缩指针，起到节约内存占用的新参数

使用-XX:+UseCompressedOops压缩对象指针

oops指的是普通对象指针(ordinary object pointers)。

Java堆中对象指针会被压缩成32位

-XX:+UseCompressedClassPointers

压缩类指针

对象中指向类元数据的指针会被压缩成32位

类指针压缩空间会有一个基地址

-XX:+UseConcMarkSweepGC

设置年老代为CMS并发收集

-XX:+UseParNewGC

设置年轻代为并行收集

附JVM参数配置示例：

```
-XX:+CMSClassUnloadingEnabled
-XX:CMSFullGCsBeforeCompaction=9
-XX:CMSInitiatingOccupancyFraction=80
-XX:+CMSParallelRemarkEnabled
-XX:+CMSScavengeBeforeRemark
-XX:+ExplicitGCInvokesConcurrent
-XX:-HeapDumpOnOutOfMemoryError
-XX:HeapDumpPath=/data/applogs/HeapDumpOnOutOfMemoryError
-XX:InitialHeapSize=5361369088
-XX:MaxHeapSize=5361369088
-XX:MaxNewSize=2061500416
-XX:MaxTenuringThreshold=9
-XX:NewSize=2061500416
-XX:OldPLABSize=16
-XX:+PrintGC
-XX:+PrintGCApplicationConcurrentTime
-XX:+PrintGCApplicationStoppedTime
-XX:+PrintGCDateStamps
-XX:+PrintGCDetails
-XX:+PrintGCTimeStamps
-XX:+PrintHeapAtGC
-XX:-ReduceInitialCardMarks
-XX:+ScavengeBeforeFullGC
-XX:SoftRefLRUPolicyMSPerMB=0
-XX:SurvivorRatio=8
-XX:ThreadStackSize=512
-XX:+UseCMSCompactAtFullCollection
-XX:+UseCMSInitiatingOccupancyOnly
-XX:+UseCompressedClassPointers
-XX:+UseCompressedOops
-XX:+UseConcMarkSweepGC
-XX:+UseParNewGC
```

题外话：

对于虚拟机参数

以 -X 开头的是非标准选项（不能保证被所有的 JVM 实现都支持），如果在后续版本的 JDK 中有变恕不另行通知。

指定 -XX 的选项是不稳定、不建议随便使用的。这些选项在今后变更恕不另行通知。

常用工具使用示例

- 1.jstack 堆栈跟踪工具

```
jstack [-l] <pid>
       (to connect to running process)
   jstack -F [-m] [-l] <pid>
       (to connect to a hung process)
   jstack [-m] [-l] <executable> <core>
       (to connect to a core file)
   jstack [-m] [-l] [server_id@]<remote server IP or hostname>
```

原文链接：JDK 1.8 Hotspot 虚拟机参数说明

```
      (to connect to a remote debug server)
Options:
    -F  to force a thread dump. Use when jstack <pid> does not respond (process is hung)
    -m  to print both java and native frames (mixed mode)
    -l  long listing. Prints additional information about locks
    -h or -help to print this help message
```

例如jstack -l 1792

- 2.jstat 统计信息监视工具

命令如下

```
jstat -<option> [-t] [-h<lines>] <vmid> [<interval> [<count>]]

Definitions:
  <option>      An option reported by the -options option
  <vmid>       Virtual Machine Identifier. A vmid takes the following form:
  <lvmid>[@<hostname>[:<port>]]
               Where <lvmid> is the local vm identifier for the target
               Java virtual machine, typically a process id; <hostname> is
               the name of the host running the target Java virtual machine;
               and <port> is the port number for the rmiregistry on the
               target host. See the jvmstat documentation for a more complete
               description of the Virtual Machine Identifier.
  <lines>      Number of samples between header lines.
  <interval>    Sampling interval. The following forms are allowed:
                  <n>["ms"|"s"]
               Where <n> is an integer and the suffix specifies the units as
               milliseconds("ms") or seconds("s"). The default units are "ms".
  <count>      Number of samples to take before terminating.
  -J<flag>     Pass <flag> directly to the runtime system.
```

jstat [-命令选项] [vmid进程的pid] [间隔时间/毫秒] [查询次数]

命令选项可选值有

- -class
- -compiler
- -gc
- -gccapacity
- -gccause
- -gcmetacapacity
- -gcnew
- -gcnewcapacity
- -gcold
- -gcoldcapacity
- -gcutil
- -printcompilation

简单示例如下：

例如jstat -gcutil 1792

- 3.jmap 内存映像工具

jmap [pid]

例如：jmap 1792

jmap -histo:live [pid] 查看当前Java进程创建的活跃对象数目和占用内存大小

例如：jmap -histo:live 1792 >vm.log

jmap -dump:live,format=b,file=heap1031.bin <pid> 导出java堆快照信息用于分析

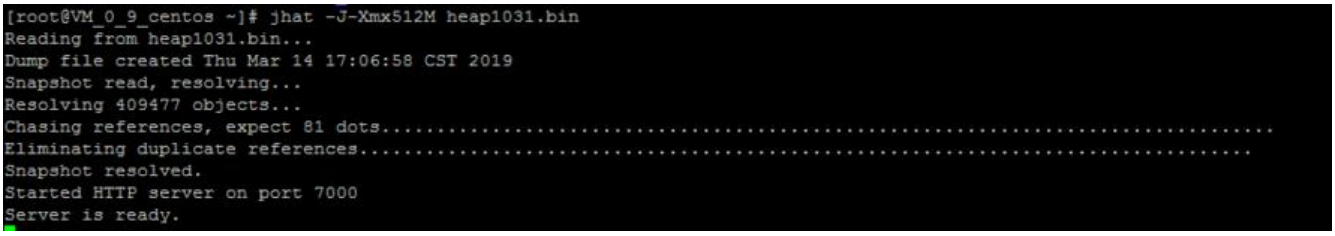
- 4.jinfo 虚拟机配置信息工具

例如:jinfo 1792

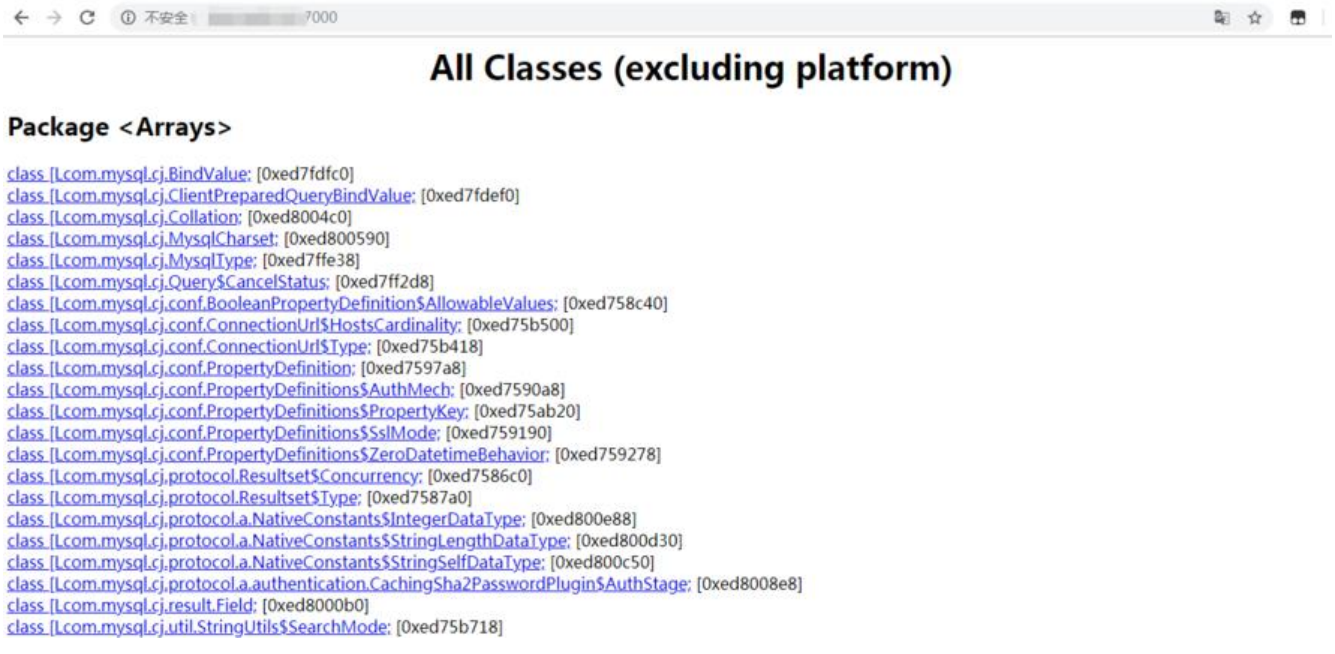
- 5.jhat 堆转储快照分析工具

分析由jmap导出的快照信息

例如:jhat -J-Xmx512M heap1031.bin



解析Java堆转储文件,并启动一个 web server,端口7000



原文链接：JDK 1.8 Hotspot 虚拟机参数说明

- 6.jconsole 监视与管理控制台
- 7.visualvm

VisualVM作为可视化监控工具，很好的监控内存、CPU、线程等资源,
jmap导出的快照也可以用visualvm来分析