# JVM常用设置参数

## JVM常用参数

### 官方推荐

根据实际事情调整新生代和幸存代的大小

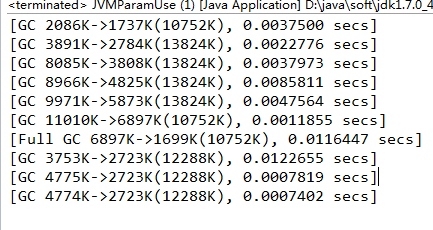
官方推荐新生代占堆的3/8

幸存代占新生代的1/10

在OOM时，记得Dump出堆，确保可以排查现场问题

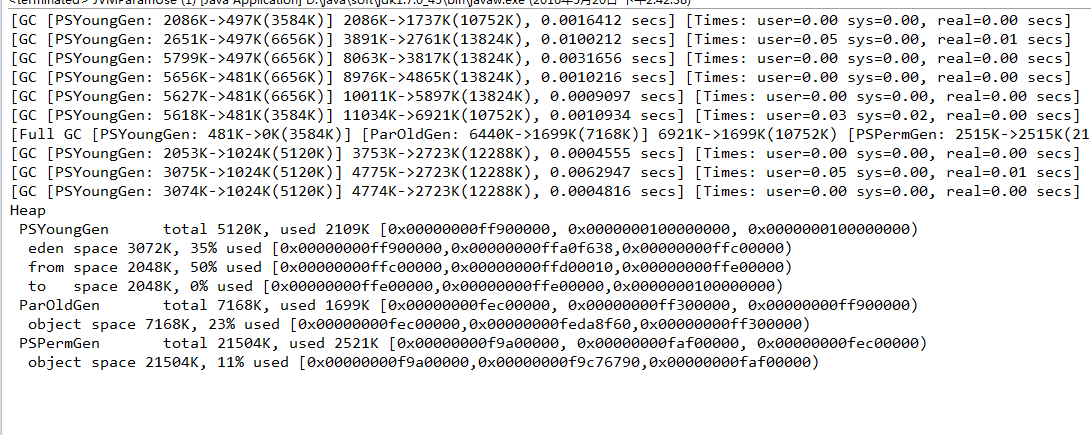
### -XX:+printGC

简要打印GC信息



### -XX:+PrintGCDetails

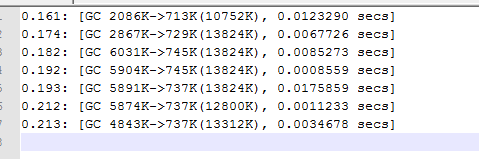
打印GC详细信息

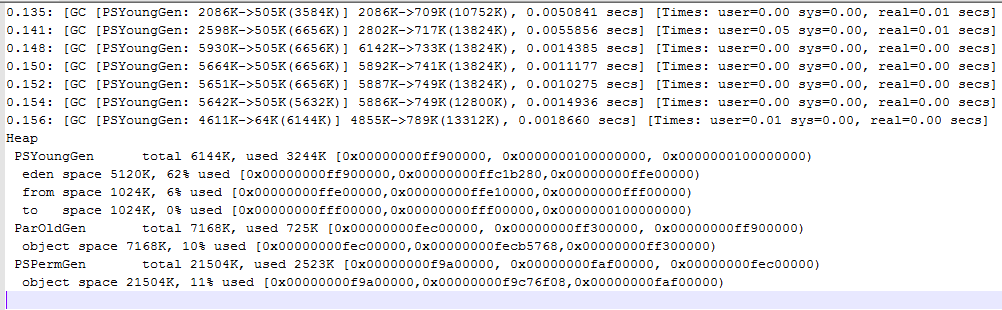


### -Xloggc:log/gc.log

指定GC log的位置(可使用相对目录)，以文件输出

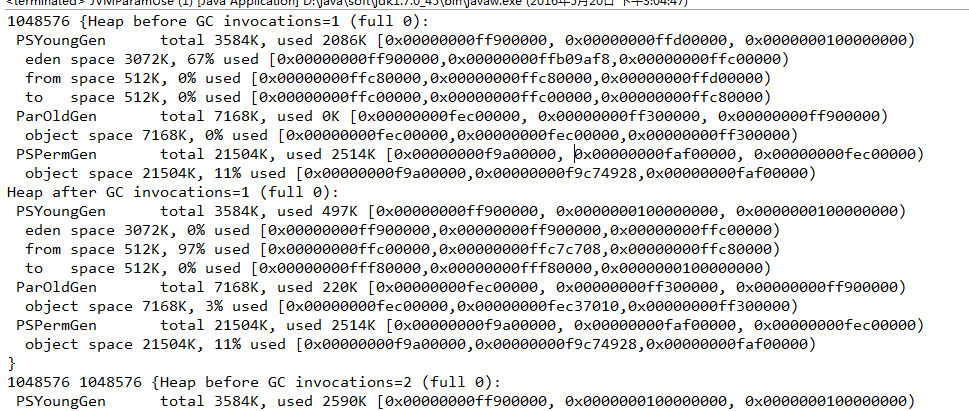
默认简要打印GC信息，可加-XX:+PrintGCDetails 输出详细日志





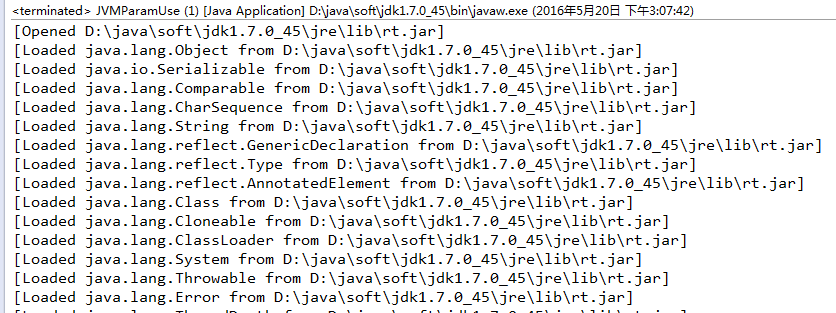
### -XX:+PrintHeapAtGC

在每一次GC后打印堆的信息



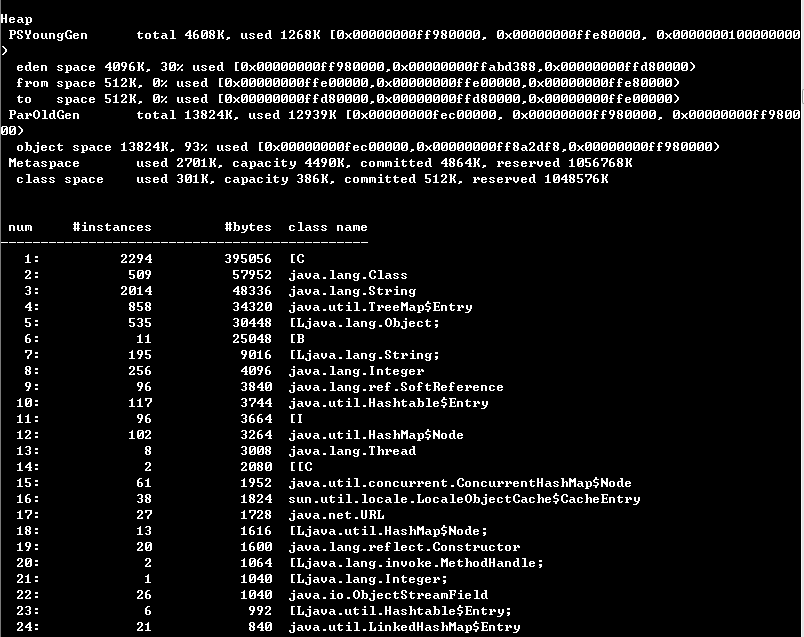
### -XX:+TraceClassLoading

监控类的加载



### -XX:+PrintClassHistogram

按下Ctrl+Break(中断键)后，打印类的信息



### -Xmx –Xms

指定最大堆和最小堆， JVM一般维持在最小堆上

### -Xmn

设置新生代大小

### -XX:NewRatio

设置新生代中(eden+2\*s)和老年代（不含永久区的区值）

4 表示新生代与老年代比值为1:4 即新生代点1/5

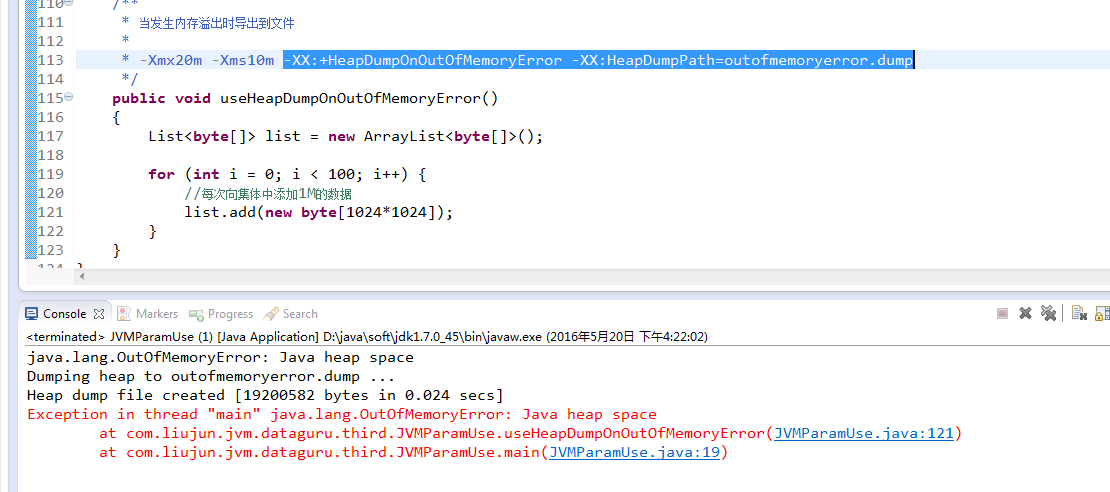
### -XX:SurvivorRatio

设置两个survivor区和Eden区的比值

如8 两个Survivor：eden区 为 2:8  即一个survivor区占1/10

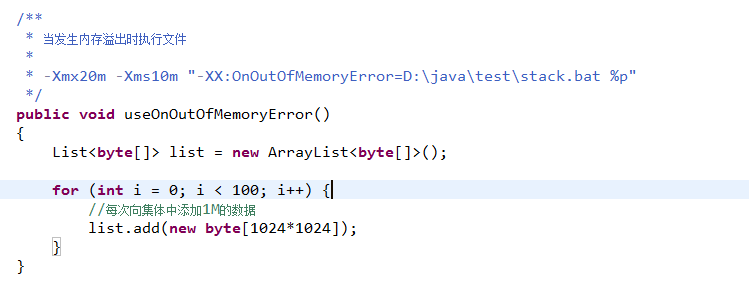
### -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=outofmemoryerror.dump

在发生内存溢出时，将内存信息写入到dump文件中去



### -XX:OnOutOfMemoryError

在OOM时，执行一个脚本



%p标识程序的pid

在发生内存溢出时，打印线程信息

stack.bat文件内容 将文件输出到指定地方

D:\java\soft\jdk1.7.0\_45\bin\jstack.exe -F %1 > D:\java\test\outofMemory.log

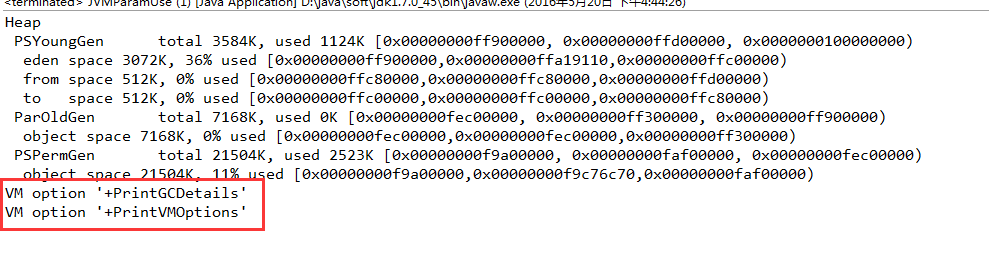
### -XX:PermSize –XX:MaxPermSize

设置永久区的初始空间和最大空间

他们表示，一个系统可以容纳多少个类型

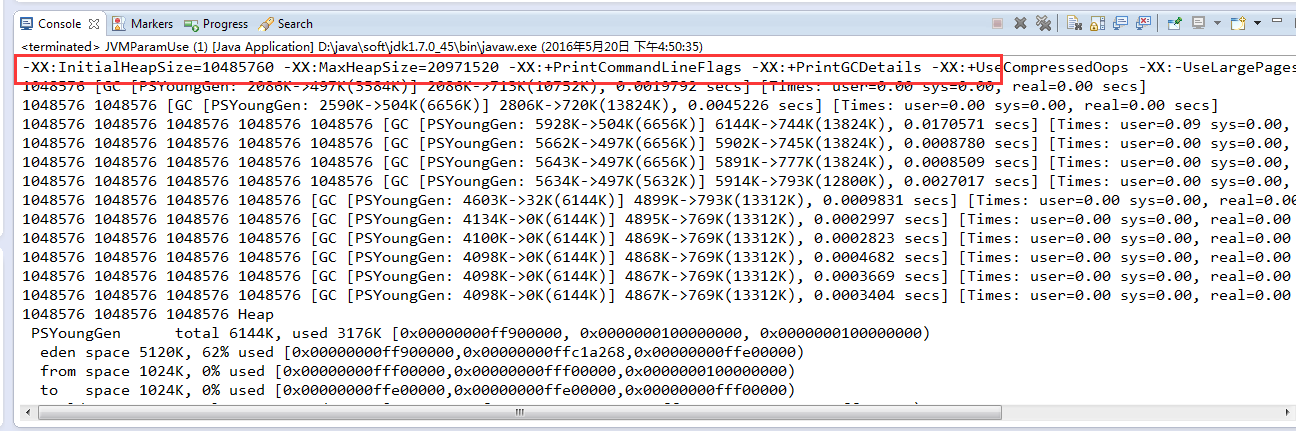
### -XX:+PrintVMOptions

打印出GC的选项信息



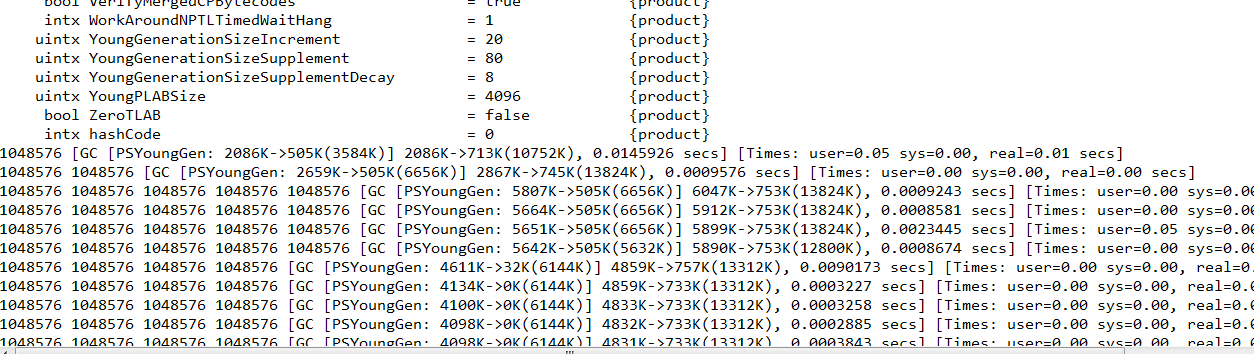
### -XX:+PrintCommandLineFlags

打印出GC的当前选项信息



### -XX:+PrintFlagsFinal

JVM支持的所有参数





## GC常用设置参数

### -XX:+UseSerialGC(串行收集器)

打开此开关后使用Serial + Serial Old的收集器组合进行内存回收

* 最古老，最稳定
* 效率高
* 可能会产生较长的停顿
* 新生代、老年代使用串行回收
* 新生代复制算法
* 老年代标记-压缩

1048576 Heap

def new generation total 3136K, used 1082K [0x00000000f9a00000, 0x00000000f9d60000, 0x00000000fa0a0000)

eden space 2816K, 38% used [0x00000000f9a00000, 0x00000000f9b0ea00, 0x00000000f9cc0000)

from space 320K, 0% used [0x00000000f9cc0000, 0x00000000f9cc0000, 0x00000000f9d10000)

to space 320K, 0% used [0x00000000f9d10000, 0x00000000f9d10000, 0x00000000f9d60000)

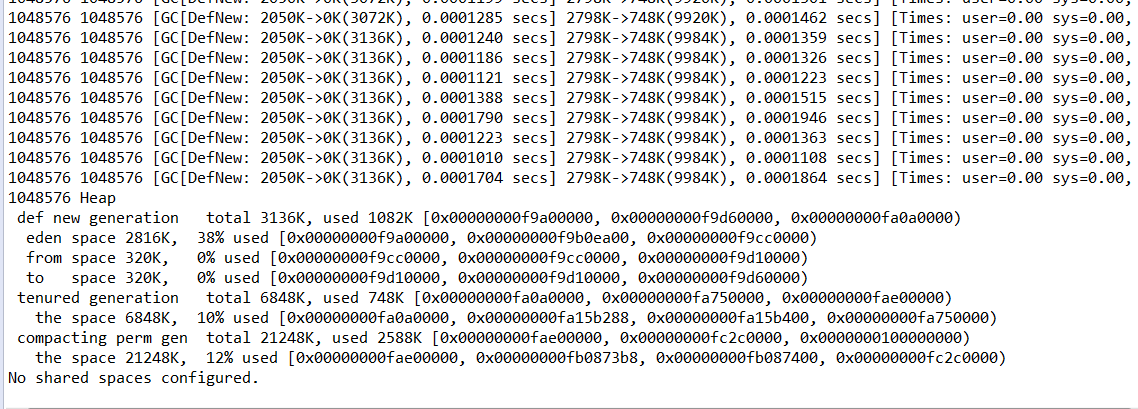
tenured generation total 6848K, used 748K [0x00000000fa0a0000, 0x00000000fa750000, 0x00000000fae00000)

the space 6848K, 10% used [0x00000000fa0a0000, 0x00000000fa15b288, 0x00000000fa15b400, 0x00000000fa750000)

compacting perm gen total 21248K, used 2588K [0x00000000fae00000, 0x00000000fc2c0000, 0x0000000100000000)

the space 21248K, 12% used [0x00000000fae00000, 0x00000000fb0873b8, 0x00000000fb087400, 0x00000000fc2c0000)

No shared spaces configured.



### -XX:+UseParNewGC(在串行收集器的基础上增加了新生代并生)

使用ParNew 与Serial Old的收集器组合进行内存回收

* 新生代并行
* 老年代串行
* 复制算法
* 多线程，需要多核支持
* -XX:ParallelGCThreads 限制线程数量

1048576 Heap

def new generation total 3136K, used 1082K [0x00000000f9a00000, 0x00000000f9d60000, 0x00000000fa0a0000)

eden space 2816K, 38% used [0x00000000f9a00000, 0x00000000f9b0ea00, 0x00000000f9cc0000)

from space 320K, 0% used [0x00000000f9cc0000, 0x00000000f9cc0000, 0x00000000f9d10000)

to space 320K, 0% used [0x00000000f9d10000, 0x00000000f9d10000, 0x00000000f9d60000)

tenured generation total 6848K, used 745K [0x00000000fa0a0000, 0x00000000fa750000, 0x00000000fae00000)

the space 6848K, 10% used [0x00000000fa0a0000, 0x00000000fa15a7e8, 0x00000000fa15a800, 0x00000000fa750000)

compacting perm gen total 21248K, used 2585K [0x00000000fae00000, 0x00000000fc2c0000, 0x0000000100000000)

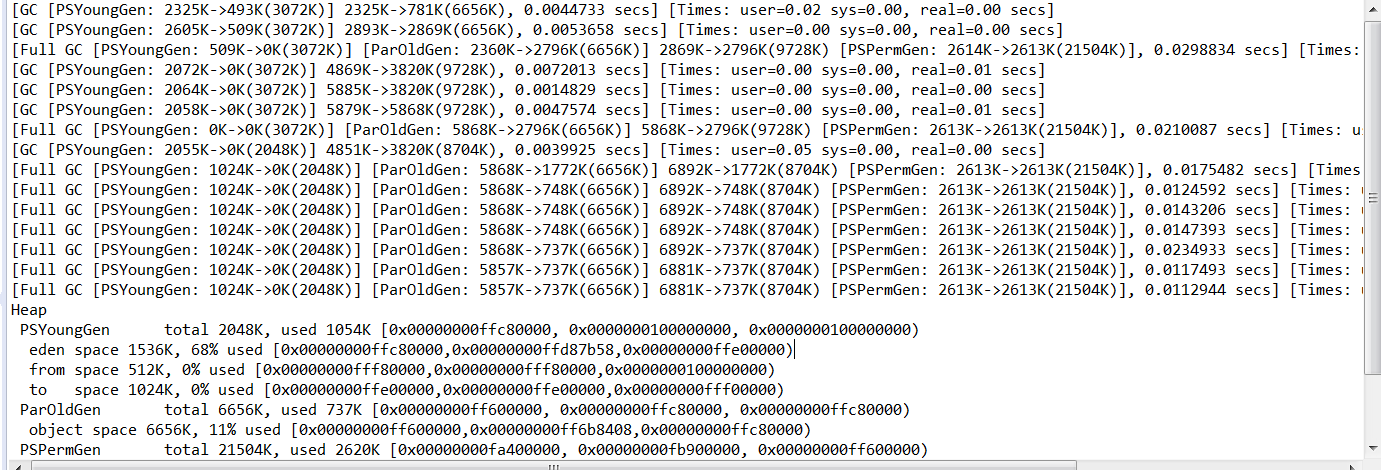
the space 21248K, 12% used [0x00000000fae00000, 0x00000000fb0865b0, 0x00000000fb086600, 0x00000000fc2c0000)

No shared spaces configured.

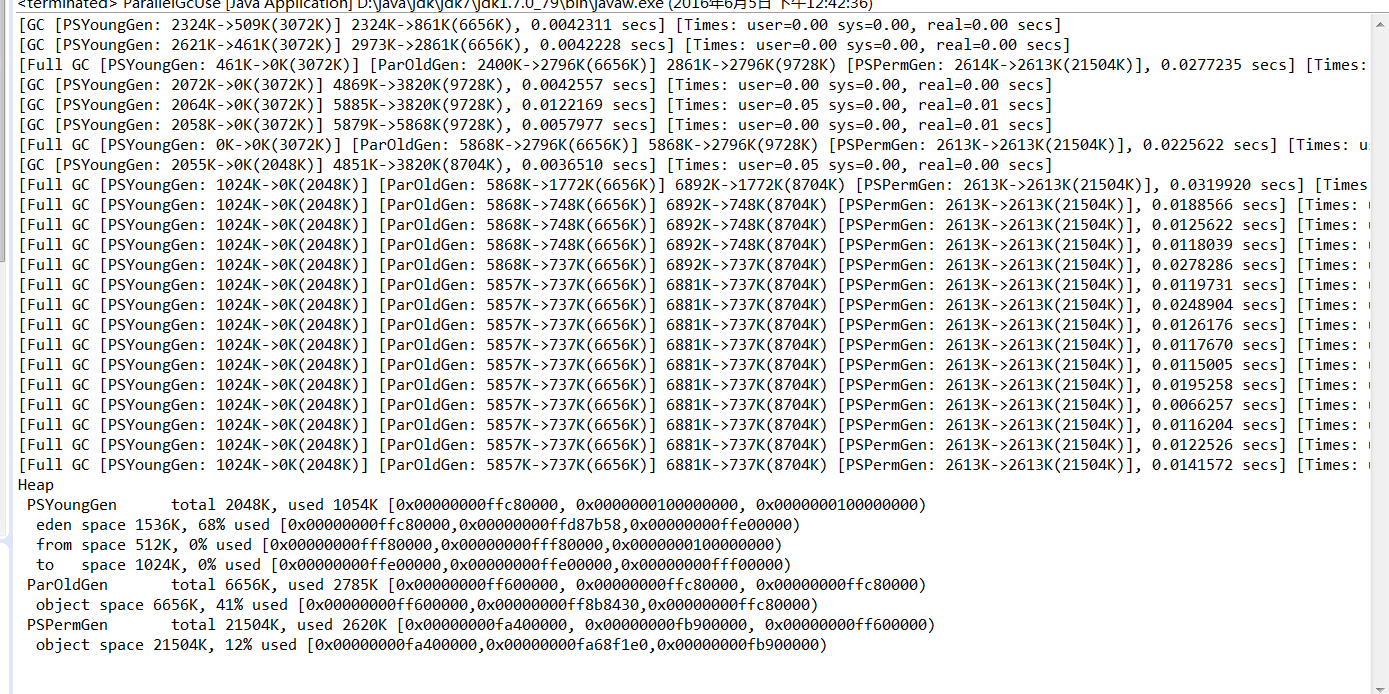
### Parallel收集器(类似ParNew收集器)

* 新生代复制算法
* 老年代 标记-压缩
* 更加关注吞吐量

#### -XX:+UseParallelGC (新生代并生化，老年代串行)



#### -XX:+UseParallelOldGC(新生代并行，老年代也并行)



#### -XX:MaxGCPauseMills,(针对parallel收集器有效)

-XX:MaxGCPauseMillis=2

最大停顿时间，单位毫秒

GC尽力保证回收时间不超过设定值

#### -XX:GCTimeRatio(针对parallel收集器有效)

0-100的取值范围

垃圾收集时间占总时间的比

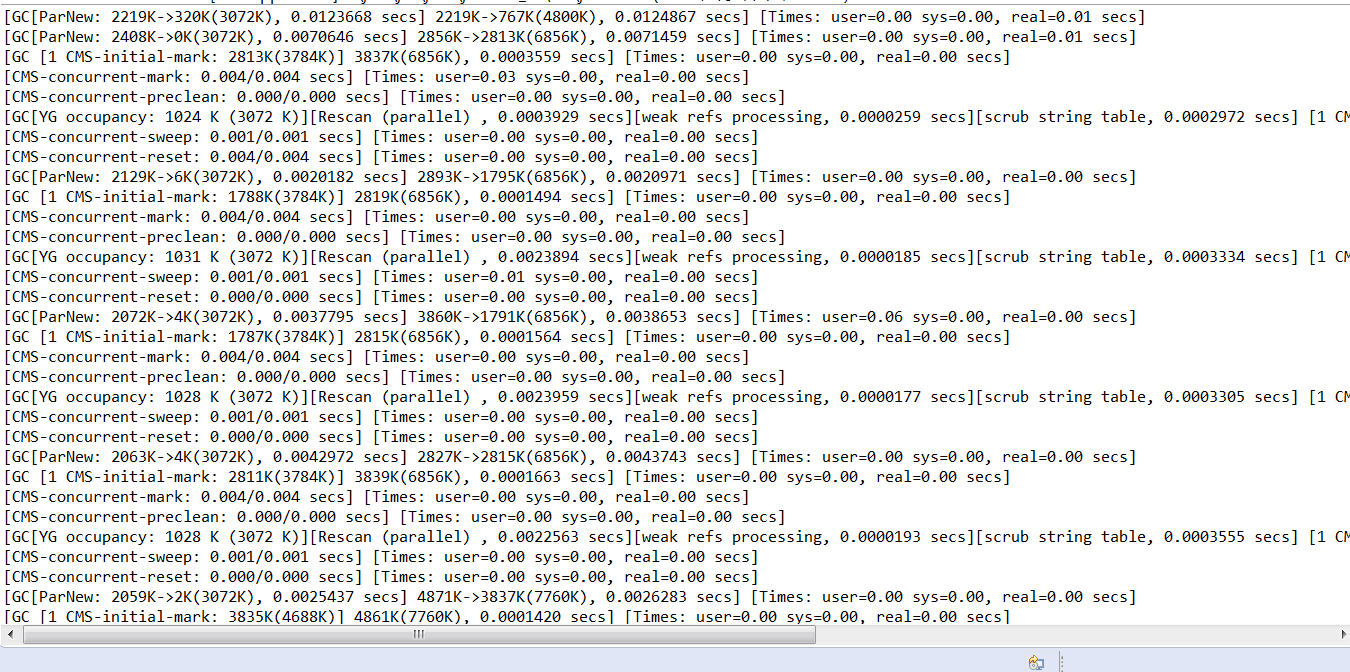
默认99，即最大允许1%时间做GC

这两个参数是矛盾的。因为停顿时间和吞吐量不可能同时调优

### CMS收集器

* Concurrent Mark Sweep 并发标记清除
* 标记-清除算法
* 与标记-压缩相比
* 并发阶段会降低吞吐量
* 老年代收集器（新生代使用ParNew）

#### -XX:+UseConcMarkSweepGC



# 内存回收策略

## 对象优先在Eden上分配

大多数情况下，对象在新生代Eden区域中进行分配，档Eden区域中没有足够的空间进行分配的时候，发起一次Minor GC

注意提到的Minor GC和Full GC

Minor GC：这就是新生代的GC，指的是新生的垃圾收集动作，由于java对象大多具有朝生夕死的特性，所以Minor GC非常频繁，一般回收速度也比较快。

Full GC：指的是发生在老年代的GC,又被称作Major GC,出现了Major GC,经常会伴随这至少一次的Minor GC，但这也不是绝对的，一般来说Major GC的速度会比Minor GC慢十倍以上。

## 大对象直接进入老年代

所谓的大对象是指大量需要连续java内存空间的java对象，最典型的大对象就是那种很长的字符串或者数组。大对象对于虚拟机分配来说是个很坏的消息，但是也有更坏的，就是那种朝生夕死的大对象，经常会出现大对象容易导致内存还有不少空间的时候就提前出发垃圾收集机制以获取足够的连续空间来“安置”他们。

虚拟机提供了一个参数，上一篇也讲到过，就是-XX:PretenureSizeThreshold参数，让大于这个参数的对象直接进入老年代分配，这样的目的就是避免在Eden区以及两个Survivor区之间发生大量的内存复制，这是因为新生代采用的复制算法，复制大量的生存对象会耗费很多资源。

## 长期存活的对象将进入老年代

虚拟机既然采用分代收集的思想来管理内存，那内存回收的时候就必须能识别那些对象放在新生代，那些对象放在老年代，为了做到这点，虚拟机给每个对象定义了一个对象年龄计数器，如果对象在Eden出生并且经过一次Minor GC后仍然存活，并且能被Survivor接纳，年龄就加一岁，当他的年龄增加到一定程度的时候，默认是15，就会晋升到老年代，这个值可以通过虚拟机的参数-XX:MaxTenuringThreshold来进行设置。

## 动态对象年龄判定

为了能更好的适应不同程序的内存情况，虚拟机并不总是要求对象的年龄必须达到MaxTenuringThreshold才能晋升老年代，如果在Survivor空间中相同年龄所有对象大小综合大于Survivor空间的一半，年龄大于等于该年龄的对象就可以直接进入老年代，无语等到MaxTenuringThreshold中要求的年龄。

## 空间分配担保：

这点就给借钱时候的保人类似，如果到时间你换不上，那么保人要先把钱换上。

言归正传，在发生Minor GC的时候，虚拟机会检测之前每次晋升到老年代的平均大小是否大于老年代的剩余空间大小，如果大于，则改为直接进行一次Full GC，如果小于，则查看HandlePromotionFailure设置是否允许担保失败；如果允许，那只会进行Minor GC，如果不允许，则也要进行一次Full GC.

由于新生代采用复制收集算法，但是为了保证内存的利用率，只是用其中一块Survivor空间作为轮换备份，因此当出现大量对象在Minor GC后任然存活的情况下，就需要老年代进行分配担保，让Survivor无法容纳的对象直接进入老年代，与生活中贷款很类似，老年代要做这样的担保，前提是老年代本身还可以容纳这些对象，但是实际中经过Minor GC之后新生代到底会有多少对象存活下来，这个是不晓得的，所以只好去之前每一次回收晋升到老年代对象大小的一个平均值作为经验值，与老年代剩余的空间大小做比较，决定是否进行Full GC来让老年代腾出更多的空间。

# 工具

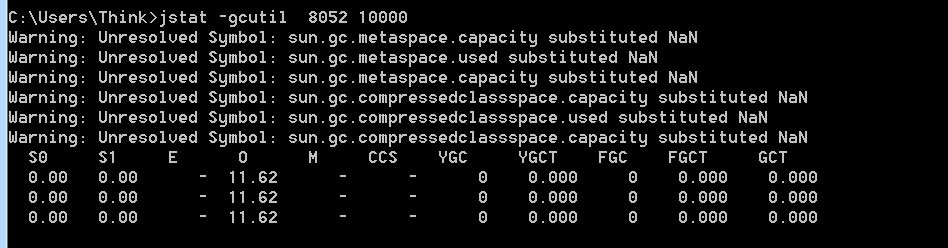
## jstat 内存查看工具

|  |  |
| --- | --- |
| **Option** | **Displays…** |
| [class](http://itzoo.info/?p=256#class_option) | class loader的行为统计。Statistics on the behavior of the class loader. |
| [compiler](http://itzoo.info/?p=256#compiler_option) | HotSpt JIT编译器行为统计。Statistics of the behavior of the HotSpot Just-in-Time compiler. |
| [gc](http://itzoo.info/?p=256#gc_option) | 垃圾回收堆的行为统计。Statistics of the behavior of the garbage collected heap. |
| [gccapacity](http://itzoo.info/?p=256#gccapacity_option) | 各个垃圾回收代容量(young,old,perm)和他们相应的空间统计。Statistics of the capacities of the generations and their corresponding spaces. |
| [gccause](http://itzoo.info/?p=256#gccause_option) | 垃圾收集统计概述（同**-gcutil**），附加最近两次垃圾回收事件的原因。Summary of garbage collection statistics (same as **-gcutil**), with the cause of the last and current (if applicable) garbage collection events. |
| [gcnew](http://itzoo.info/?p=256#gcnew_option) | 新生代行为统计。Statistics of the behavior of the new generation. |
| [gcnewcapacity](http://itzoo.info/?p=256#gcnewcapacity_option) | 新生代与其相应的内存空间的统计。Statistics of the sizes of the new generations and its corresponding spaces. |
| [gcold](http://itzoo.info/?p=256#gcold_option) | 年老代和永生代行为统计。Statistics of the behavior of the old and permanent generations. |
| [gcoldcapacity](http://itzoo.info/?p=256#gcoldcapacity_option) | 年老代行为统计。Statistics of the sizes of the old generation. |
| [gcpermcapacity](http://itzoo.info/?p=256#gcpermcapacity_option) | 永生代行为统计。Statistics of the sizes of the permanent generation. |
| [gcutil](http://itzoo.info/?p=256#gcutil_option) | 垃圾回收统计概述。Summary of garbage collection statistics. |
| [printcompilation](http://itzoo.info/?p=256#printcompilation_option) | HotSpot编译方法统计。HotSpot compilation method statistics. |

jstat -gcutil 8052 10000

8052为程序Pid

这个命令是每个10秒钟输出一次jvm的gc信息，10000指的是间隔时间为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，所以实际执行的时候，会有很多行）：



例如

 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

S0:新生代的susvivor0区，空间使用率为54..62%

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，空间使用率43.52%

P:持久带，空间使用率86.24%

YGC:minor gc执行次数1792次

YGCT:minor gc耗费的时间5.093毫秒

FGC:full gc执行次数33

FGCT:full gc耗费的时间7.670毫秒

GCT:gc耗费的总时间12.763毫秒