JVM优化

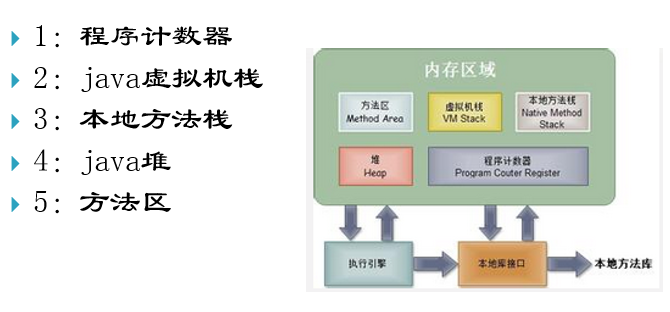
# Java虚拟机

一个运行时的java虚拟机实例的天职是负责运行一个java程序，在启动一个java程序的同时会诞生一个虚拟机实例，当该程序退出时，虚拟机实例也随之消亡，如果在同一台计算机上同时运行三个java程序，会得到三个java虚拟机实例，每个java程序都运行于它自己的java虚拟机实例中。

java虚拟机内部有两个线程：守护线程与非守护线程，守护线程通常由虚拟机内部使用，例如执行垃圾收集任务，而java的初始线程为非守护线程

当非守护线程终止后，虚拟机实例将自动退出当然也可以通过System.exit方法退出。

## 内存模型



### 程序计数器

是一块很小内存空间，它的作用可以看做是当前线程所执行的字节码的行号指示器。字节码解释工作时就是通过改变这个计数器的值来选取下一条需要执行的字节码指令，分支，循环，跳转，异常处理等基础功能都由计数器所完成。

### java虚拟机栈

也是线程私有的内存空间，它和java线程在同一时间创建，它保存方法的局部变量、部分结果，并参与方法的调用和返回。每一个方法被调用直至完成的过程，就对应着一个栈帧在虚拟机栈中从入栈到出栈的过程。

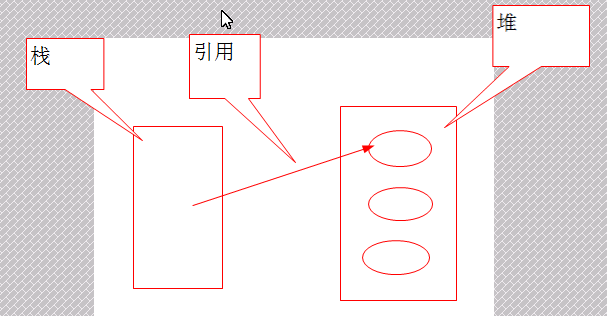
java虚拟机规范中，定义了两种异常与栈空间相关：StackOverflowError及OutOfMemoryError. 使用-Xss参数来设置栈的大小，栈的大小直接决定了函数调用可以达的深度

由于栈中包括局部变量表，操作数栈，返回地址等信息，方法在调用时，如果方法的参数和局部变量相对较多，那么栈中的局部变量表就越大，它的栈帧就越大。栈深度者越浅。

### 本地方法栈

和java虚拟机栈的功能很相似，java虚拟机栈用于管理java函数的调用，而本地方法栈用于管理本地方法的调用。本地方法并不是java实现的而由C来实现

### java堆



是java运行时内存中最为重要部分，**几乎所有的对象和数组**都是在堆中分配空间。 java堆可以分为新生代和老年代两个部分，新生代用于存放刚刚产生的对象和年轻对象，如果对象一直没有被回收，生存得足够长，新生代对象就会被移入到老年代。

新生代包括：Eden区 s0区 s1区（s0与s1为survivor空间）

老年代只有一个区,老年代是存放时间较长，经过垃圾回收次数较多的对象。

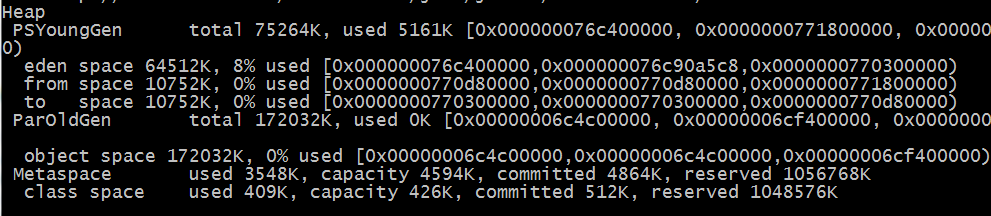
老年代

S0

S1

Eden

java -XX:+PrintGCDetails Test3



### 方法区

也是JVM内存区中非常重要的一块内存区域，与堆空间类似，它是被JVM中所有的线程共享，方法区主要保存的信息是类的元数据。

方法区中最重要的是类的类型信息，常量池，域信息，方法信息等。

方法区也称为永久区，是一块独立于java堆的内存空间，但GC也能回收，通常通过二个方面回收

1：对常量池回收 2：对元数据回收。

## jvm内存分配参数

### 设置最大堆内存

java应用程序可以使用的最大堆可以用-Xmx参数指定，最大堆指的是新生代和老年代的大小之和的最大值。

### 设置最小堆内存

java应用程序可以使用的最小堆可以用-Xms参数指定，也就是JVM启动时，所占据的操作系统内存的大小。

java程序在运行时 首先会被分配-Xms指定的内存大小，并尽可能尝试在这个空间段内运行程序，确实无法满足时才会向操作系统申请更多的内存，直到内存大小达到-Xmx指定的最大内存为止，如果超过-Xmx的值，则抛出OutOfMemoryError

### 设置新生代

参数-Xmn用于设置新生代的大小，设置一个较大的新生代会减少老年代的大小，这个参数的堆系统性能以及GC行为有很大的影响，新生代的大小一般设置为整个堆空间的1/3.

### 设置持久代

持久代（方法区）不属于堆的一部分，使用-XX:MaxPermSize可以设置持久代的最大值，使用-XX:PermSize可以设置持久代的初始大小。

持久代的大小决定了系统可以支持多少个类定义和多少常量。

一般来说MaxPermSize设置为64MB已经可以满足绝大部分的程序正常工作，如果依然出现溢出，可以设置为128MB。

主要用于大量的动态类生成.当需要生成大量的动态类时需要设置。

### 设置线程栈

线程栈是线程一块私有空间，在JVM中可以使用-Xss参数设置线程栈的大小。-Xss设置越大 线程数越小，且与堆大小有关。

如果系统需要大量的线程并发执行，那么设置一个较小的堆和较小的栈，有助于提高系统所能承受的最大线程数

### 堆的比例分配

JVM初始分配的内存由-Xms指定，JVM最大分配的内存由-Xmx指定。默认空余堆内存小于40%时，JVM就会增大堆直到-Xmx的最大限制；空余堆内存大于70%时，JVM会减少堆直到 -Xms的最小限制。因此服务器一般设置-Xms、-Xmx相等以避免在每次GC 后调整堆的大小。

-XX:NewRatio= 参数可以设置Young与Old的大小比例，-server时默认为1:2,如果太小，会使大对象直接分配到old区去，增大major collections的执行的次数，影响性能。

-XX:SurvivorRatio= 参数可以设置Eden与Survivor的比例，默认为1:8，Survivio大了会浪费，如果小了的话，会使一些大对象在做minor gc时，直接从eden区潜逃到old区，让old区的gc频繁。这个参数保持默认就好了，一般情况下，对性能影响不大。

启动后可通过jmap –heap [pid]查看。

由于堆的整体大小是固定的，young generation越大，tenured generation越小，越会增加major collections的执行的次数。所以最佳的选择是由对象的生命周期分布所决定。

## 垃圾收集

### 垃圾收集器的类型

按线程数分，可以分为串行垃圾回收器和并行垃圾回收器。串行垃圾回收器一次只使用一个线程进行垃圾回收；并行垃圾回收器一次将开启多个线程同时进行垃圾回收。

按工作模式：分为并发垃圾回收器及独占垃圾回收器

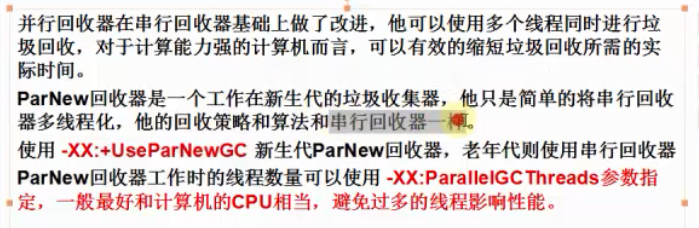
按碎片处理：压缩垃圾回收器及非压缩垃圾回收器

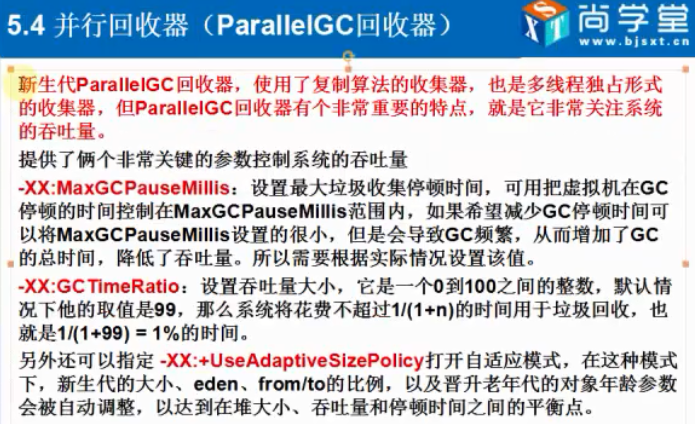
按代分：新生代垃圾回收器及老年代垃圾回收器。

### 回收器的选择

1：新生代并行收集器：

-XX：+UseParNewGC





2: 新生代和老年代都用的并行回收器

-XX：+UseParallelOldGC

3: 新生代和老年代都用的串行回收器

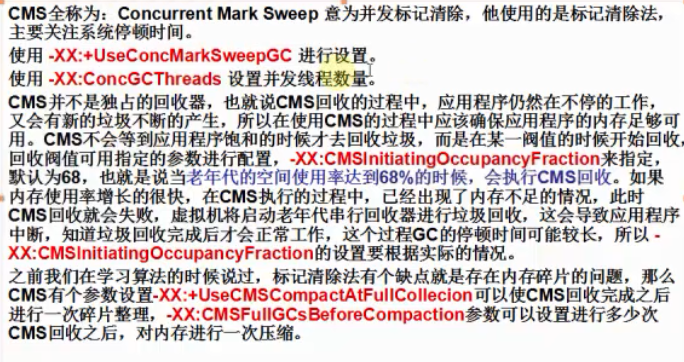
-XX:+UseSerialGC

4：CMS回收器

-XX:+UseConcMarkSweepGC

当生产环境中，系统莫名其妙地停止工作，有可能是由于垃圾回收引起的。

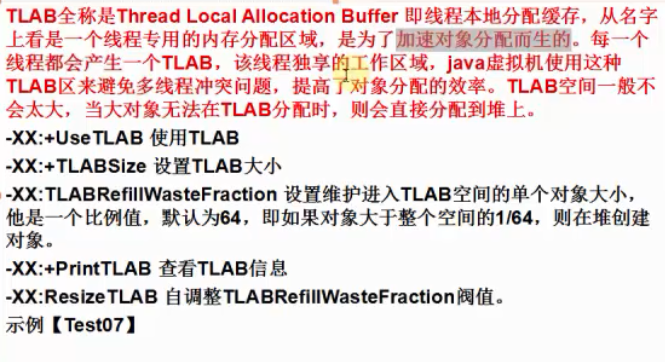
选择串行收集器



java -Xmx512M -Xms512M -XX:+UseSerialGC -Xloggc:gc.log -XX:+PrintGCDetails StopWorldTest

选择G1收集器 并设置停顿时间不超过10ms

java -Xmx512M -Xms512M -XX:+UseG1GC -XX:MaxGCPauseMillis=10 -Xloggc:gc.log -XX:+PrintGCDetails StopWorldTest



CMS收集器

G1收集器

GC调试案例

GC相关参数

## 调优方法

### 新对象预留新生代

由于fullGC的成本要远远高于MinorGC,因此尽可能将对象分配在新生代是明智的。大部分情况下JVM会尝试在eden区分配对象，但由于空间紧张等，很可能不得不将部分年轻对象向老年代压缩，所以调优时，可以为应用程序分配一个合理的新生代空间，以最大限度避免新对象直接进入老年代。

-XX:+PrintGCDetails -Xmx20M -Xms20M 与-XX:+PrintGCDetails -Xmx20M -Xms20M -Xmn6M比较

### 大对象进入老年代

将大对象直接分配到老年代，保持新生代对象的结构的完整性，以提高GC效率。

-XX:+PrintGCDetails -Xmx20M -Xms20M 与 -XX:+PrintGCDetails -Xmx20M -Xms20M -XX:PretenureSizeThreshold=1000000 进行比较观察出大对象分配给老年代

-XX:PretenureSizeThreshold：设置大对象直接进入老年代的阈值，当对象的大小超过这个值，将直接在老年代分配.

### 稳定与震荡的堆大小

稳定的堆大小是对垃圾回收有利的，获得一个稳定的堆大小的方法是使-Xms和-Xmx的大小一致。

### 吞吐量优先

吞吐量优先的方案会尽可能减少系统的执行垃圾回收的总时间，故可以考虑关注系统吞吐量的并行回收收集器。

对于4G内存 32核CPU的计算机上

使用以下参数设置：

1：-Xmx3800M -Xms3800M

2: -Xss128k :减少线程栈大小

3：-Xmn2G :设置新生代大小

4：-XX:+UseParallelGC:新生代并行回收器，这是一个关注吞吐量的收集器，可以尽量减少GC的时间

5：-XX:+ParallelGCThreads=20:设置用于垃圾回收的线程数，通常情况下可以和CPU数量相等。但CPU较多的情况下，设置相对较小的数值也是合理的

6：-XX:+UseParallelOldGC:老年代也使用并行回收收集器。

### 降低停顿

为了降低应用软件在垃圾回收的停顿，首先考虑的是使用关注停顿的CMS回收器，其次，为了减少FullGC次数，应尽量将对象预留在新生代，因为新生代的MinorGC成本远低于老年代的FullGC

可以如下设置降低停顿

1：-XX:ParalleGCThreads:设置20个线程进行垃圾回收

2：-XX:UseParNewGC:新生代使用并行回收器

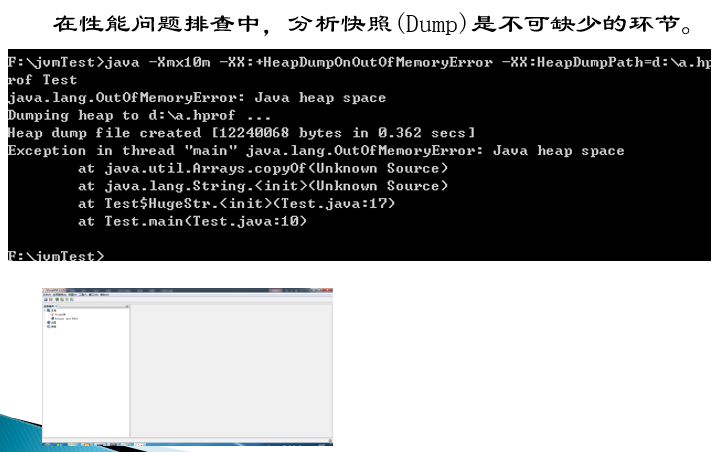
3：-XX:+UseConcMarkSweepGC：使用CMS收集器降低停顿

4：-XX:SurvivorRatio=8 设置eden区和survivor区的比例为8:1

5：-XX:TargetSurvivorRatio=90:设置survivor使用率为90%

6：-XX:MaxTenuringThreshold=30：年轻对象到老年对象的年龄默认是15次，也就是15次minorGC依然存活，故设置为30，即 尽量让对象保存在新生代

### 快照堆



然后用jvisualvm.exe 载入。

### 获取GC信息

-verbose:gc 或-XX:+PrintGC都能获取GC信息

-XX:+PrintGCDetails：获取更加详细的信息。

-XX:+PrintGCTimeStamps:获取GC的频率和间隔

-XX:+PrintHeapAtGC:获取堆的使用情况

-Xloggc:D:\gc.log:指定日志情况

### Tomcat 调优

Tomcat启动加速

1: 添加GC日志

set CATALINA\_OPTS=%CATALINA\_OPTS% "-Xloggc:gc.log"

2：设置堆大小

set CATALINA\_OPTS=%CATALINA\_OPTS% "-Xmx32M"

set CATALINA\_OPTS=%CATALINA\_OPTS% "-Xms32M"

3: 设置禁用GC

set CATALINA\_OPTS=%CATALINA\_OPTS% "-XX:+DisableExplicitGC"

4: 扩大新生代的大小

set CATALINA\_OPTS=%CATALINA\_OPTS% "-XX:NewRatio=2"

5: 禁止JVM安全校验

set CATALINA\_OPTS=%CATALINA\_OPTS% "-Xverify:none"

6: 禁止用元数据回收

set CATALINA\_OPTS=%CATALINA\_OPTS% "-Xnoclassgc"

set CATALINA\_OPTS=%CATALINA\_OPTS% "-Xloggc:gc.log"

set CATALINA\_OPTS=%CATALINA\_OPTS% "-Xmx32M"

set CATALINA\_OPTS=%CATALINA\_OPTS% "-Xms32M"

set CATALINA\_OPTS=%CATALINA\_OPTS% "-XX:+DisableExplicitGC"

set CATALINA\_OPTS=%CATALINA\_OPTS% "-XX:NewRatio=2"

set CATALINA\_OPTS=%CATALINA\_OPTS% "-Xverify:none"

set CATALINA\_OPTS=%CATALINA\_OPTS% "-Xnoclassgc"

set CATALINA\_OPTS=%CATALINA\_OPTS% "-XX:+UseConcMarkSweepGC"