### Java虚拟机

#### 一、Java内存区域

##### 1.内存区域分配及其作用：

线程共享

**堆（Heap）：**内存中最大的一块，存放对象实例和数组,堆内存在物理上可以不连续，通过-Xms、-Xmx可以设置堆内存大小。

-Xms:堆内存初始值，默认物理内存1/64

-Xmx:堆内存最大值，默认物理内存1/4

-Xmn:堆内存新生代大小

**方法区(Method Area/Non-Heap)：**存放类信息、常量、静态变量（static）等，通过-XX:MetaspaceSize -XX:MaxMetaspaceSize设置方法区内存大小。

线程独立

**栈(JVM Stacks)：**描述的是java方法执行的内存模型，每个方法被执行的时候都会创建一个栈帧（是一种数据结构），用于存储局部变量表（存储的基础数据类型和对象引用）、操作栈（可用于运算）、动态链接（对符号引用进行解析和链接的过程）、方法出口等信息。每个方法被调用直至执行完成的过程，就对应一个栈帧在虚拟机栈中从入栈到出栈的过程。通过-Xss设置栈内存大小。

**本地方法栈(Native Method Stacks)：**作用同栈，与native方法相关

**程序计数器(Program Counter Register)：**用于指示当前线程代码执行的位置

##### 2.OutOfMemoryError（OOM）异常

**Java性能分析工具**：JProfiler（idea插件、软件下载同时下载，需要破解）

**场景一**：堆内存溢出，堆转储快照文件

JVM设置:设置堆初始内存20m,最大内存20m,报OOM异常时hprof文件存储在桌面aa文件夹中。

-Xms20m -Xmx20m -XX:+HeapDumpOnOutOfMemoryError

-XX:HeapDumpPath=C:\Users\Administrator\Desktop\aa

**场景二**：栈、本地方法栈溢出，异常类型StackOverflowError、OutOfMemoryError。超过虚拟机允许的最大栈深度（最大栈帧、栈中的最大方法数）会抛StackOverflowError

JVM设置：-Xss128k 设置栈内存为128kb

**场景三：**方法区内存溢出

JVM设置：-XX:MetaspaceSize=3m -XX:MaxMetaspaceSize=3m 设置方法区内存初始值为3m，最大值为3m。注：PermSize在jdk1.8后弃用

#### 二、垃圾回收

对于线程独立的内存区域，如：栈、本地方法栈、程序计数器，内存随着线程的创建而分配，随着线程的结束而回收。这部分区域不是垃圾回收的重点，垃圾回收器关注的是堆、方法区的内存回收！

##### 1.对象是否存活

方式一：引用计数算法（循环引用问题）

方式二：根搜索算法（GC Roots）,主流商用语言使用的算法

当一个对象到GC Roots中没有任何引用链时，则该对象不可用。

**2.GC Roots对象包含：**

1.栈帧中局部变量表中的引用对象

2.方法区类静态属性引用的对象

3.方法区常量引用的对象

4.本地方法栈的JNI(native方法)引用的对象

**3.引用的类型：**

1.强引用：Strong Reference如Object object = new Object,只要强引用存在，就不会被垃圾回收器回收。

2.软引用：Soft Reference 系统将要发生内存溢出前，会将这些对象做一次垃圾回收。

3.弱引用：Weak Reference 弱引用对象在下一次垃圾回收时会被回收。

4.虚引用：Phantom Reference 该对象被回收时会收到系统通知。

**4.垃圾回收算法**

1.标记-清除算法：

Mark-Sweep,标记需要回收的对象，统一清除回收。缺点：效率不高，会产生大量不连续的内存碎片。

2.复制算法：

Copying 将内存按容量分为相等的两块，每次只使用其中的一块，当内存满了时候，将活着的对象复制到另外一块内存中，再对内存一次清除，这样解决了产生大量不连续内存碎片的问题，但是内存的使用率不高。

3.标记-整理算法：

Mark-Compact 标记过程通标记-清除算法，在回收之前，让所有存活的对象都往一端移动，这样就避免了产生大量不连续内存碎片的问题。

4.分代收集算法：当前商业虚拟机采用的算法

Generational Collection 根据对象的存活周期将内存划分为几块，一般是把Java堆内存分为新生代和年老代，新生代采用复制算法，年老代采用标记-整理或标记-清除算法。

**5.垃圾收集器**

垃圾回收算法的具体实现！

**JVM命令**：

java -XX:+PrintCommandLineFlags –version cmd/linux中可查看虚拟机配置。

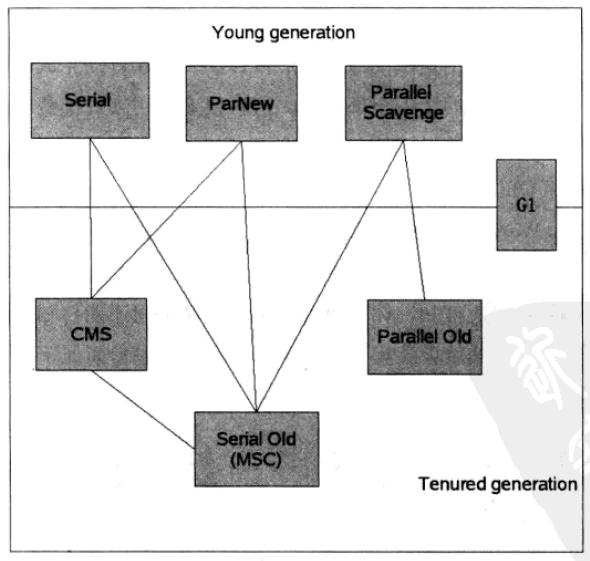
-XX:+PrintGCDetails：GC时查看内存使用情况

-XX:SurvivorRatio：设置年轻代内存比例大小（一般Eden:survivor:survivor=8:1:1）

-XX:PretenureSizeThreshold：配置对象直接进入年老代的大小 (该配置只对Serial和ParNew收集器有效)

-XX:MaxTenuringThreshold：配置年龄计数器的默认值，该值是对象进入年老代的阈值。

**垃圾回收器的配合使用：**



**新生代收集器**

1.Serial收集器

单线程收集器，一个线程完成垃圾回收工作。GC线程工作时，其他线程（包括用户线程）会被暂停。简单高效。采用复制算法。

2.ParNew收集器

Serial收集器的多线程版。多个线程进行垃圾回收工作。采用复制算法。

3.Paraller Scanvenge收集器

吞吐量优先收集器，该垃圾收集器适用于新生代，采用标记复制算法、多线程模型进行垃圾收集。与其他新生代垃圾收集器的差别是，它更关注于吞吐量，而不是停顿时间。一般来说，需要与用户交互的程序更关注较短的停顿时间，而如果是需要达成尽量大的吞吐量的话，则该处理器会更加适合。其通过-XX：UseAdaptiveSizePolicy参数，可以开启其自动调节功能，适用于对垃圾收集器的调优不太了解的用户。

**年老代收集器**

1.Serial Old 收集器

Serial收集器的老年代版本，采用标记-整理算法，是一个单线程收集器。

2.Paraller Old 收集器

Paraller Scanvenge的老年代版本，采用标记-整理算法，多线程收集器。

3.CMS 收集器

Concurrent Mark Sweep,是一种获取最短回收停顿时间为目标的收集器，注重服务的响应速度，给用户带来更好的体验，采用的是标记-清除算法。

**通用收集器**

G1收集器

Garbage First 较新的一种垃圾回收器。采用的是标记-整理算法

**6.内存分配策略**

永久代在JDK1.8后弃用，改为元数据空间Metaspace



1.对象优先在Eden区分配

大多数情况，对象在新生代Eden区分配。当Eden区没有足够的空间时，虚拟机将发起一次Minor GC。

-Xms20m -Xmx20m -Xmn10m -XX:SurvivorRatio=8 -XX:+PrintGCDetails

2.大对象直接进入老年代

-Xms20m -Xmx20m -Xmn10m -XX:SurvivorRatio=8 -XX:+PrintGCDetails

-XX:PretenureSizeThreshold=3000000(该配置只对Serial和ParNew收集器有效)

3.长期存活的对象将进入老年代

虚拟机给每个对象定义了一个年龄计数器 age,每经过一次Minor GC任然存活，age增长1，当年龄增长到15时（默认），就会晋升到老年代。这个阈值可以设定：

-XX:MaxTenuringThreshold

4.动态对象年龄判断

Survivor空间中相同年龄的所有对象大小（内存大小）的总和大于Survivor空间的一般，年龄大于等于该年龄的对象就可以进入老年代。

5.空间分配担保

发生Minor GC时，虚拟机会检测之前每次晋升到年老代的对象大小的平均值是否大于年老代剩余的空间大小，如果大于，则直接改为一次Full GC。如果小于，对象晋升到年老代。HandlePromotionFailure用于设置是否允许担保失败，允许则进行Minor GC，否则进行Full GC。

**7.Minor GC和Full GC**

Minor GC指的是新生代GC,指发生在新生代的垃圾回收动作，Java对象大多具有朝生夕灭的特性，所以Minor GC非常频繁，一般回收速度也比较快。

Full GC/Major GC指的是老年代GC,指发生在老年代的垃圾回收动作，Full GC的速度一般比Minor GC慢10倍左右。

#### 三、性能监控

#### 四、性能调优

#### 五、类加载机制

#### 六、程序编译与代码优化

#### 七、高效并发