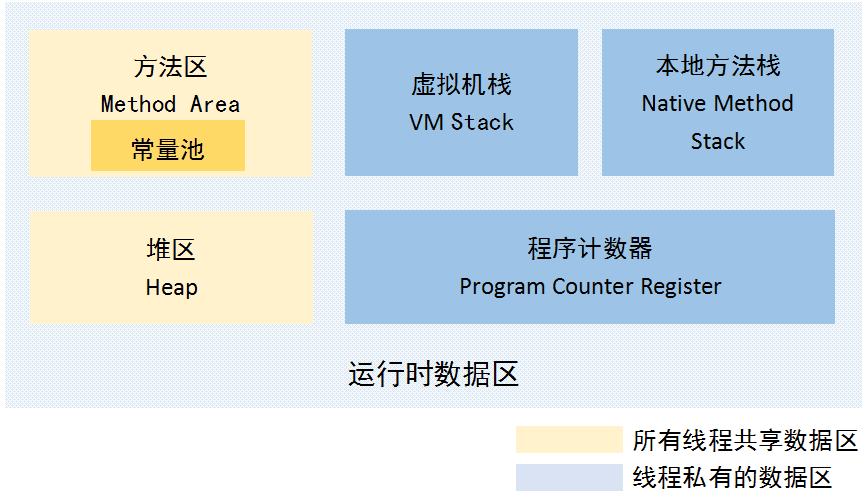
# JAVA内存

## java内存模型

### 在java程序运行时内存主要分为五个区域,分别为方法区(1.8中已经被matespace代替,使用本地直接内存),虚拟机栈,本地方法栈,堆区,程序计数器.



## jvm内存配置

-Xms 初始堆大小。如：-Xms256m

-Xmx 最大堆大小。如：-Xmx512m

-Xmn 新生代大小。通常为 Xmx 的 1/3 或 1/4。新生代 = Eden + 2 个 Survivor 空间。实际可用空间为 = Eden + 1 个 Survivor，即 90%

-Xss JDK1.5+ 每个线程堆栈大小为 1M，一般来说如果栈不是很深的话， 1M 是绝对够用了的。

-XX:NewRatio 新生代与老年代的比例，如 –XX:NewRatio=2，则新生代占整个堆空间的1/3，老年代占2/3 即新生代是老年代的几分之几大小,若 Xms = Xmx, 并且设定了 Xmn, 那么该项配置就不需要配置了

-XX:SurvivorRatio 新生代中 Eden 与 Survivor 的比值。默认值为 8.则Eden占8/10,两个survivor各占1/10, 但这个值表示一个Survivor是Eden的几分之几,如果为4 则Eden占4/6,两个survivor各占1/6

-XX:PermSize 永久代(方法区)的初始大小(1.8后失效,在1.8后设置会忽略并警告)

-XX:MaxPermSize 永久代(方法区)的最大值(1.8后失效, 在1.8后设置会忽略并警告)

-XX:MetaspaceSize  设置元空间初始大小(1.8后生效)

-XX:MaxMetaspaceSize=128m 设置元空间最大大小(1.8后生效)

-XX:+PrintGCDetails 打印 GC 信息

-XX:+HeapDumpOnOutOfMemoryError 让虚拟机在发生内存溢出时 Dump 出当前的内存堆转储快照，以便分析用

-XX:PretenureSizeThreshold 当对象大于多少(byte)时对象直接进入老年代

# GC

# GCLog

## 打印GCLog的jvm启动参数

-XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+PrintGCTimeStamps

-Xloggc:gc.log 将gc日志打印到文件中

### 打印的GCLog详细说明

2018-06-12T15:04:46.496+0800: 1160.388: [GC (Allocation Failure) [PSYoungGen: 147488K->8736K(144896K)] 160339K->21595K(320000K), 0.0045867 secs] [Times: user=0.00 sys=0.00, real=0.00 secs]

1. gc发生的时间
2. gc发生距启动的时间(s)
3. gc发生的原因(Allocation Failure无法在新生代中给将创建的对象分配内存)
4. gc回收内存方式(PSYoungGen 采用并行垃圾回收算法对新生代进行回收),回收前新生代占用堆内存->回收后占用堆内存(垃圾回收后新生代可用堆内存)
5. jvm总的堆内存占用回收前后变化 前->后(总可用堆内存)
6. GC事件的持续时间(Duration),单位是秒
7. GC事件的持续时间,通过多种分类来进行衡量:

user – 此次垃圾回收, 垃圾收集线程消耗的所有CPU时间(Total CPU time).

sys – 操作系统调用(OS call) 以及等待系统事件的时间(waiting for system event)

real – 应用程序暂停的时间(Clock time). 由于串行垃圾收集器(Serial Garbage Collector)只会使用单个线程, 所以 real time 等于 user 以及 system time 的总和.

## Minor GC 和 Full GC

### Minor GC

又称新生代GC，指发生在新生代的垃圾收集动作；

因为Java对象大多是朝生夕灭，所以Minor GC非常频繁，一般回收速度也比较快；

### Full GC

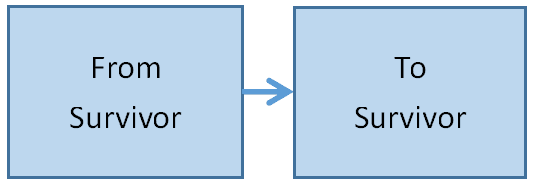
又称**Major GC或老年代GC**，指发生在老年代的GC；

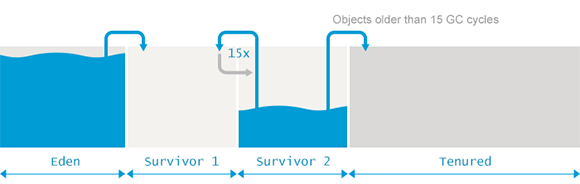
出现Full GC经常会伴随至少一次的Minor GC（不是绝对，Parallel Sacvenge收集器就可以选择设置Major GC策略）；

Major GC速度一般比Minor GC慢10倍以上；

## GC原理

GC发生的主要原因是为了内存的回收利用,由于在程序运行的过程中会产生很多临时的对象,而这些临时对象占据了运行时内存的大部分空间,如果不及时清理这些无用对象,就会造成内存溢出,而java提供了关于这一部分对象的垃圾回收机制,这样在程序中就无需手动释放内存.由于jvm的栈区主要保存方法运行时的数据,方法调用结束内存就会释放,所以GC发生在jvm的堆区和方法区,主要是jvm的堆区,在hotspot中对于堆的回收方式为分代回收,堆中主要分为新生代和老年代,新生代分为 eden,survive1,survive2.所有在堆中分配空间的对象都优先在新生代中分配空间,当新生代中无多余空间分配时,就进行一次MinorGC ,此时将eden中的对象进行回收,将仍然存活的且能被survive容纳(超出survive大小的直接进入老年代)的对象放入survive中,survive主要作为新生代到老年代的缓冲,在以后的gc中仍然存活的对象会在两个survive中来回移动,起始的survive称为from survive,结束的称为to survive,移动过后,from survive被清空.对象在survive经过一定次数的互相移动即经过一定次数的MinorGC,表明对象生存周期的持久性,最后存活对象得以进入老年代.







当要创建的对象需要分配的空间过于巨大,即使新生代进行一次MinorGC后也无法容纳该对象,则该对象直接在老年代分配空间进行创建.如果老年代也无法容纳该对象,则进行一次MajorGC即FullGC,尝试腾出更多空间,如果MajorGC后空间还是不足则抛出OutOfMemeryException.

## GC收集器

### 参考:<深入理解Java虚拟机:JVM高级特性与最佳实践>3.4

Serial收集器

最基本,历史最悠久的收集器,使用一条线程进行垃圾回收,回收时会停止所有用户线程.是虚拟机运行在Client模式下的新生代默认收集器(虚拟机默认以Server模式运行)

ParNew收集器

是Serial收集器的多线程版本,使用多条线程进行垃圾回收,在多cpu的环境中会有比Serial更好的效率,只有它与Serial能与老年代收集器CMS(Concurrent Mark Sweep)配合工作,当启用-XX:+UserConcMarkSweepGC选项后是默认的新生代收集器.通过-XX:+UseParNewGC强制使用作为新生代收集器.

Parallel Scavenge(HotSpot默认新生代收集器)

这是一个并行的垃圾回收器,主要关注程序运行时的吞吐量,吞吐量=运行用户代码的时间/(运行用户代码的时间+垃圾回收的时间),要吞吐量大,则垃圾回收时间要尽量小,所以就要求大量的垃圾一次性的回收,程序一次性停止较长的时间,但总的垃圾回收时间消耗较小,适合不需要太多交互的程序.

配置启动参数

-XX:MaxGCPauseMillis 最大GC停止时间(ms) 值过小则会减小新生代大小,会增加GC次数.

-XX:GCTimeRatio 垃圾回收时间占用户代码时间的几分之几

-XX:+UseAdaptiveSizePolicy 开关型参数,当打开后 虚拟机会根据系统运行情况对虚拟机的各种如新生代大小,Eden与Survivor的比例,晋升老年代的年龄,进行动态的调整.用以提供合适的停顿时间或最大的吞吐量.

Serial Old收集器

Serial收集器的老年代版本,主要用于虚拟机在Client模式下进行老年代的收集,

Parallel Old收集器

是ParallerScavenge的老年代版本,可以和ParallerScavenge组合使用,在注重吞吐量和CPU资源敏感的场所可以优先烤炉,ParallerScavenge加ParallerOld收集器.

GMS收集器

是一种以获取最短回收停顿时间为目标的收集器,重视服务器的响应速度

G1收集器

参考文章https://tech.meituan.com/g1.html

新一代的垃圾收集器,能对新生代和老年代进行垃圾收集, G1 GC是一个响应时间优先的GC算法，它与CMS最大的不同是，用户可以设定整个GC过程的期望停顿时间，参数-XX:MaxGCPauseMillis指定一个G1收集过程目标停顿时间，默认值200ms.启动参数-XX:+UseG1GC.