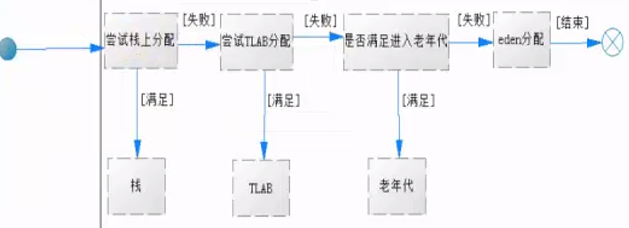
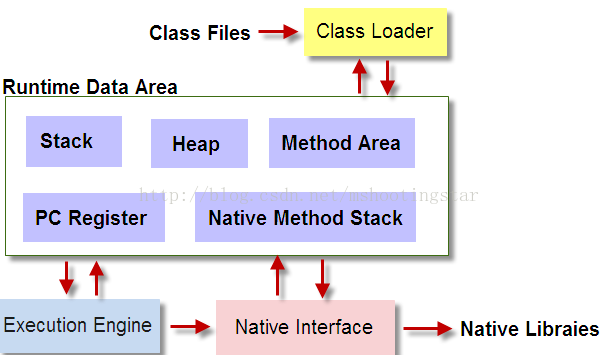
并发编程学习笔记

# JVM

1. 基本结构：



|  |  |
| --- | --- |
|  |  |
| ClassLoader | 负责加载class文件，class文件在文件开头有特定的文件标示 |
| Native Interface | 调用不同语言的本地接口给JAVA用，他会在Native Method Stack中记录对应的本地方法，然后调用该方法时就通过Execution Engine加载对应的本地lib |
| Execution Engine | Class文件被加载后，会把指令和数据信息放入内存中，它负责把这些命令解释给操作系统。 |
| Java栈 | 每个线程独有，每次线程执行到一个新的方法时就在栈里面压入一个栈帧。  帧里包含了方法里的局部变量，操作数栈以及帧数据区 |
| 本地方法栈 | Java栈是执行Java方法的线程申请的内存，而本地方法是执行本地方法而申请的内存 |
| java堆 |  |
| 方法区 |  |
| 垃圾回收系统 |  |
| 总结 | java堆和方法区是所有线程共享的，Java栈和本地方法栈以及PC寄存器都是线程独占的  NonHeap包含PermGen和Code Cache，PermGen包含Method Area  java8中PermGen已经从JVM中移除并被MetaSpace取代 |

|  |  |
| --- | --- |
| OutOfMemoryError | StackOverflowError |
| private static void oom()  {  ArrayList<Integer> s=new ArrayList<>();  while(true) {  s.add(1);  }  } | private static void stackof()  {  double s;  stackof();  } |

JVM内存分配策略

对象内存主要分配在新生代Eden区, 如果启用了本地线程分配缓冲, 则优先在TLAB上分配, 少数情况能会直接分配在老年代, 或被拆分成标量类型在栈上分配(JIT优化). 分配的规则并不是百分百固定, 细节主要取决于垃圾收集器组合, 以及VM内存相关的参数.

即使在**可达性分析算法**中不可达的对象, VM也并不是马上对其回收, 因为要真正宣告一个对象死亡, 至少要经历两次标记过程: 第一次是在可达性分析后发现没有与GC Roots相连接的引用链, 第二次是GC对在**F-Queue**执行队列中的对象进行的小规模标记(对象需要覆盖finalize()方法且没被调用过).

垃圾收集算法

分代收集算法：根据对象存活周期的不同将内存划分为几块, 如JVM中的新生代、老年代、永久代. 这样就可以根据各年代特点分别采用最适当的GC算法：新生代-复制算法；老年代-标记清除算法；老年代-标记整理算法；永久代-方法区回收

分区收集算法：将整个堆空间划分为连续的不同小区间, 每个小区间独立使用, 独立回收.可以控制一次回收多少个小区间.

垃圾收集器组合：

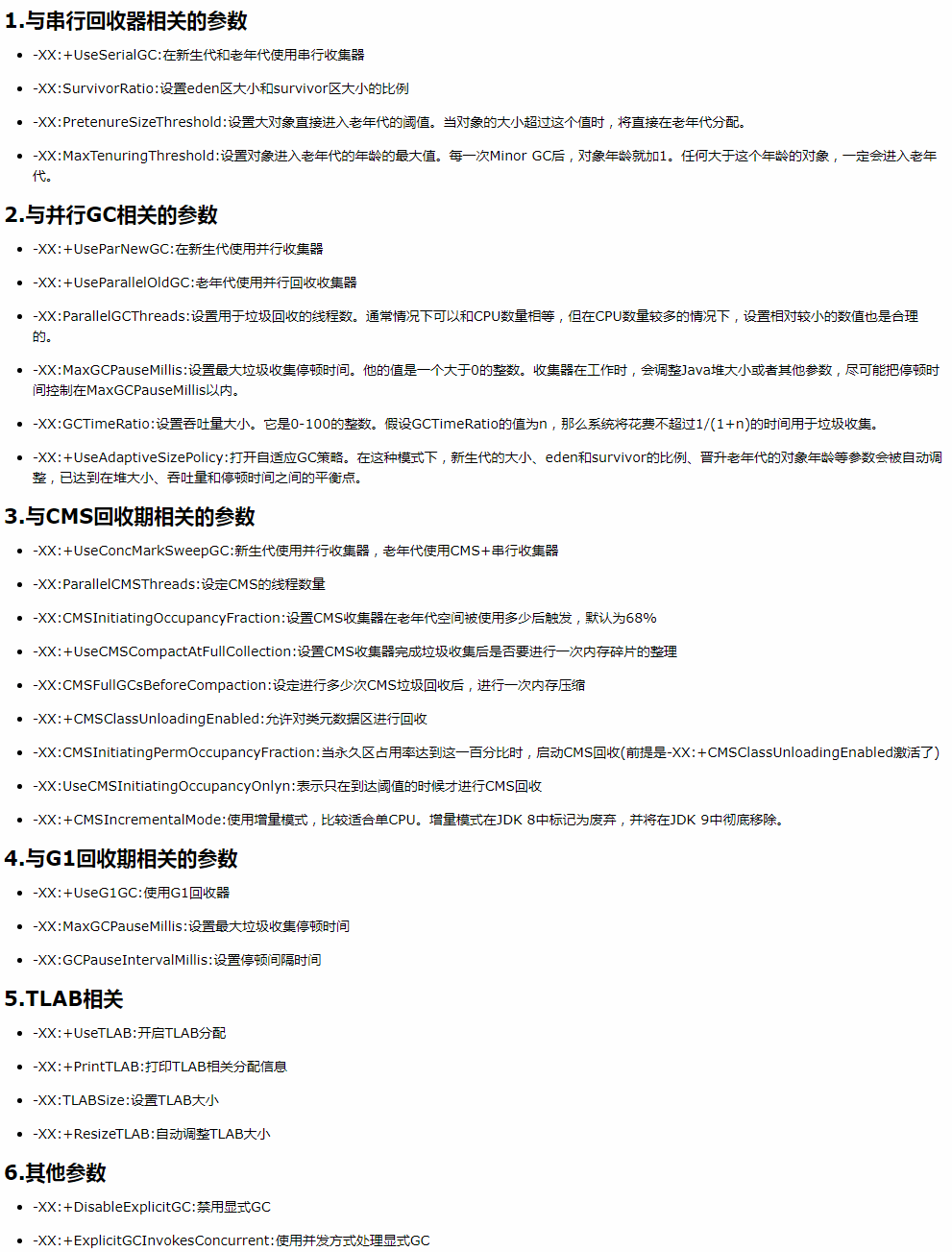
Serial/Serial Old

ParNew/Serial Old：与上边相比，只是比年轻代多了多线程垃圾回收而已

ParNew/CMS：当下比较高效的组合

Parallel Scavenge/Parallel Old：自动管理的组合

G1：最先进的收集器，但是需要JDK1.7update14以上



# ActiveMQ

JMS