java虚拟机管理大内存 PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/222/2021_2022_java_E8_99_9A_E6_8B_9F_c104_222332.htm 众所周知，jvm的内存是受限的，一为机器的体系架构，二为操作系统本身。x86,x86-64,SPARC,.....的内存映射是不同，而各操作系统的内存管理机制也有区别。以下是来自http://fengyouhua.javaeye.com/blog/58170 1. Heap设定与垃圾回收Java Heap分为3个区，Young，Old和Permanent。Young保存刚实例化的对象。当该区被填满时，GC会将对象移到Old区。Permanent区则负责保存反射对象，本文不讨论该区。JVM的Heap分配可以使用-X参数设定，-Xms初始Heap大小-Xmxjava heap最大值 -Xmnyoung generation的heap大小JVM有2个GC线程。第一个线程负责回收Heap的Young区。第二个线程在Heap不足时，遍历Heap，将Young 区升级为Older区。Older区的大小等于-Xmx减去-Xmn，不能将-Xms的值设的过大，因为第二个线程被迫运行会降低JVM的性能。为什么一些程序频繁发生GC？有如下原因：l 程序内调用了System.gc()或Runtime.gc()。l 一些中间件软件调用自己的GC方法，此时需要设置参数禁止这些GC。l Java的Heap太小，一般默认的Heap值都很小。l 频繁实例化对象，Release对象。此时尽量保存并重用对象，例如使用StringBuffer()和String()。 如果你发现每次GC后，Heap的剩余空间会是总空间的50%，这表示你的Heap处于健康状态。许多Server端的Java程序每次GC后最好能有65%的剩余空间。经验之谈：1．Server端JVM最好将-Xms和-Xmx设为相同值。为了优化GC

，最好让-Xmn值约等于-Xmx的1/3[2]。2．一个GUI程序最好是每10到20秒间运行一次GC，每次在半秒之内完成[2]。注意：1．增加Heap的大小虽然会降低GC的频率，但也增加了每次GC的时间。并且GC运行时，所有的用户线程将暂停，也就是GC期间，Java应用程序不做任何工作。2．Heap大小并不决定进程的内存使用量。进程的内存使用量要大于-Xmx定义的值，因为Java为其他任务分配内存，例如每个线程的Stack等。2．Stack的设定每个线程都有他自己的Stack。-Xss每个线程的Stack大小Stack的大小限制着线程的数量。如果Stack过大就好导致内存溢漏。-Xss参数决定Stack大小，例如-Xss1024K。如果Stack太小，也会导致Stack溢漏。3．硬件环境硬件环境也影响GC的效率，例如机器的种类，内存，swap空间，和CPU的数量。如果你的程序需要频繁创建很多transient对象，会导致JVM频繁GC。这种情况你可以增加机器的内存，来减少Swap空间的使用[2]。4．4种GC第一种为单线程GC，也是默认的GC。，该GC适用于单CPU机器。第二种为Throughput GC，是多线程的GC，适用于多CPU，使用大量线程的程序。第二种GC与第一种GC相似，不同在于GC在收集Young区是多线程的，但在Old区和第一种一样，仍然采用单线程。-XX: UseParallelGC参数启动该GC。第三种为Concurrent Low Pause GC，类似于第一种，适用于多CPU，并要求缩短因GC造成程序停滞的时间。这种GC可以在Old区的回收同时，运行应用程序。-XX: UseConcMarkSweepGC参数启动该GC。第四种为Incremental Low Pause GC，适用于要求缩短因GC造成程序停滞的时间。这种GC可以在Young区回收的同时，回收一部分Old区对象。-Xincgc参数启动该GC。