**一、**

-Xint, -Xcomp, -Xmixed

-Xint 强制全部使用字节码解释运行，-Xcomp全部编译成机器码执行，-Xmixed是混合模式，jdk默认的启动参数。

**二、**

1.jvm参数分类

Jvm参数分为三类。第一类包括了标准参数，java –help即可显示；第二类是X参数，可以用java –X来检索；第三类是XX参数，所有的XX都以“-XX:”开始，随后语法取决于参数类型：

1）对于布尔类型的参数，用“+”或者“-”，-XX:+<name>用于激活<name>选项，-XX:-<name>用于注销<name>选项。

2）对于非布尔类型的参数，需要赋值，-XX:<name>=<value>

-XX:+PrintCompilation 可以简单输出一些字节码转化成本地代码的编译过程。

-XX:+CITime 输出编译时间。

-XX:+CITime和-XX:+PrintCompilation一起使用可以对JIT编译器有更好的理解，如果使用-Xint，那么编译时间显示是0，因为是字节码解释运行，不需要编译。

2.jvm参数调优

-XX:+PrintFlagsFinal -XX:+PrintFlagsInitial 可以输出所有XX参数和值。

-XX:+PrintCommandLineFlags 打印出所有用户或者jvm默认设置过的XX参数和值。

-Xms和-Xmx（等同于-XX:InitialHeapSize and -XX:MaxHeapSize），默认和最大堆内存大小。

-XX:+HeapDumpOnOutOfMemoryError 让JVM在发生内存溢出时候自动生成堆内存快照，默认快照的存放路径在JVM启动目录下java\_pid<pid>.hprof文件里（<pid>指JVM进程的进程号），也可以通过设置-XX:HeapDumpPath=<path>改变默认堆内存快照生成路径。

-XX:OnOutOfMemoryError 利用该命令，可以在发生内存溢出时，执行一些清理工作等。

|  |
| --- |
| $ java -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/tmp/heapdump.hprof -XX:OnOutOfMemoryError ="sh ~/cleanup.sh" MyApp  该例子表示当发生内存溢出时，堆内存快照写到/tmp/heapdump.hprof路径下， |

-XX:PermSize and -XX:MaxPermSize 设置永久代的初始大小和最大值。注意这里设置的永久代不会被包括在使用参数-Xmx（-XX:MaxHeapSize）设置的堆内存大小中。

-XX:InitialCodeCacheSize and -XX:ReservedCodeCacheSize，用来存储已编译方法生成的本地代码，如果被占满，那么JIT编译器将被停用。

-XX:NewSize，-XX:MaxNewSize 设置新生代的大小

-XX:NewRatio，设置老年代和新生代比例，优点是新生代会随着整个堆大小动态扩展。

|  |
| --- |
| $ java -XX:NewSize=32m -XX:MaxNewSize=512m -XX:NewRatio=3 MyApp  JVM会尝试为新生代分配1/4的堆空间，但是不会小于32M，也不会大于512M |

-XX:SurvivorRatio，伊甸园区和幸存区大小比例。比如-XX:SurvivorRatio=10，表示一点远去是幸存区to的比例是10，所以eden：survivor1：survivor2 = 10：1：1。

-XX:+PrintTenuringDistribution，执行JVM在每次新生代GC时，输出幸存区中对象的年龄分布。

**三、吞吐量收集器**

评估一个垃圾收集算法的两个标准：

1.吞吐量越高算法越好

2.暂停时间越短算法越好

“高吞吐量”和”低暂停时间”是相互矛盾的。应用程序在GC期间会停止，这会引起额外线程调度开销：直接开销是上下文切换，间接开销是缓存的影响。如果程序运行期间只进行一次开销，那么会有高吞吐量，但是GC时会有很长的暂停时间；如果频繁GC，那么会有低的暂停时间，但是会降低吞吐量。

**Hotspot垃圾收集**

下面是面向吞吐量垃圾收集算法有关的JVM配置参数：

1. -XX:+UseSerialGC，串行垃圾收集器，年轻代和老年代都只有一个线程执行垃圾回收，适用于单个处理器的JVM

2. -XX:+UseParallelGC，JVM使用多线程并行执行年轻代垃圾收集。在java7中，-XX:+UseParallelGC和-XX:+UseParallelOldGC 一样的效果

3. -XX:+UseParallelOldGC，激活年轻代和老年代都进行并行垃圾收集。当想要高吞吐量，并且JVM有两个及以上可以处理器核心时，推荐用这个。

-XX:ParallelGCThreads，可以通过-XX:ParallelGCThreads=<value>指定并行垃圾收集的线程数量，如果不指定就使用默认的，取决于java Runtime.availableProcessors()返回的数量N，如果N<=8,并行垃圾收集器将使用N个垃圾收集线程，如果N>8，垃圾收集线程数量应该为3+5N/8。

-XX:GCTimeRatio，通过-XX:GCTimeRatio=<value>设置JVM吞吐量达到的目标值，即程序线程执行时间与总的程序执行时间达到N/(N+1)的比值。默认是99，即吞吐量是99%。

-XX:MaxGCPauseMillis，通过-XX:MaxGCPauseMillis=<value>告诉JVM最大暂停时间的目标值（毫秒）。最大暂停时间比最大吞吐量优先级要高。注意不要设置最大暂停时间太小，不然JVM需要增加GC次数，会严重影响吞吐量。

**四、CMS收集器**

并发标记清理收集器（CMS收集器）主要目标：低应用停顿时间。

包括：

初始标记：需要stop the world，标记GC Roots能直接关联的对象

并发标记：根据初始标记获取的对象引用，遍历所有对象引用

重新标记：需要stop the world，修正标记的对象记录

并发清除：进行垃圾回收

CMS收集器需要调优的两点：

1.堆碎片

CMS收集器没有碎片整理机制，如果没有连续的空间，会发生full gc。

2.对象分配率高

如果获取对象实例的频率高于收集器清除的频率，当没有足够的空间，会发生full gc

**与CMS收集器相关联的调优参数**：

-XX：+UseConcMarkSweepGC，激活CMS收集器，默认收集器。

-XX：UseParNewGC，当使用CMS收集器时，激活年轻代使用多线程并行执行垃圾回收。当使用CMS时，该配置自动开启。

-XX：+CMSConcurrentMTEnabled，并发的CMS阶段将以多线程执行。

-XX：ConcGCThreads，-XX：ConcGCThreads=<value>定义并发CMS过程运行的线程数，如果没设置，JVM会根据-XX：ParallelGCThreads设置的并行垃圾手机线程数来计算默认CMS并发线程数，公式：ConcGCThreads=（ParallelGCThreads+3）/4

-XX:CMSInitiatingOccupancyFraction，-XX:CMSInitiatingOccupancyFraction=<value>代表老年代堆空间使用率阈值，当到达value时，会触发CMS垃圾回收，默认是68。

-XX:+CMSClassUnloadingEnabled，CMS收集器默认不会对永久代进行垃圾回收，当永久代空间占满，会用full gc的方式进行垃圾回收，配置该参数以后，可以让CMS对永久代进行垃圾回收。

-XX:+ExplicitGCInvokesConcurrent 和 -XX:+ExplicitGCInvokesConcurrentAndUnloadsClasses，如果使用CMS收集器，当系统中调用System.gc()时，会触发full gc，-XX:+ExplicitGCInvokesConcurrent可以令无论什么时候调用系统gc，都执行CMS GC，而不是full gc。-XX:+ExplicitGCInvokesConcurrentAndUnloadsClasses保证了当有系统GC调用时，永久代也包括在内。

-XX:+DisableExplicitGC，忽略系统调用的GC。

**五、GC日志**

-XX:+PrintGC，开启简单GC日志模式，为每一次新生代GC和每一次Full GC打印一行信息。

-XX:PrintGCDetails，详细GC日志模式。

-XX:+PrintGCTimeStamps和-XX:+PrintGCDateStamps，将时间和日期加入到GC日志中，这两个参数可以同时使用。