JVM参数及调优

调优基本概念

在调整性能时，JVM有三个组件：堆大小调整，垃圾收集器调整，JIT编译器。大多数调优选项都与调整堆大小和为你的情况选择最合适的垃圾收集器有关。JTI编译器对性能也有很大影响，但很少需要使用较新版本的JVM进行调优。

通常，在调优Java应用程序时，重点是一下两个主要目标之一：

1. 响应性：应用程序或系统对请求的数据进行响应的速度，对于专注于响应性的应用程序，长的暂停时间是不可接受的，重点是在短时间内做出回应。
2. 吞吐量：侧重在特定时间内最大化应用程序的工作量，对于专注于吞吐量的应用程序，高暂停时间是可接受的。由于高吞吐量应用程序在较长时间内专注于基准测试，因此不需要考虑快速响应时间。

常用的JVM参数

-XX:+AlwaysPreTouch	jvm启动时分配内存，非使用时再分配
-XX：ErrorFile = filename	崩溃日志
-XX:+TraceClassLoading	跟踪类加载信息
-XX:+PrintClassHistogram	按下Ctrl+Break后，打印类的信息
-Xmx –Xms	最大堆和最小堆
-xx:permSize、-xx:metaspaceSize	永久代 /元数据空间
-XX:+HeapDumpOnOutOfMemoryError	OOM时导出堆到文件
-XX:+HeapDumpPath	OOM时堆导出的路径
-XX:OnOutOfMemoryError	在OOM时，执行一个脚本
java -XX:+PrintFlagsFinal –version 打印所有的-XX参数和默认值	

GC调优思路

1. 分析场景：例如：启动速度慢；偶尔出现响应慢于平均水平或者出现卡顿。
2. 确定目标：内存占用，低延时，吞吐量。
3. 收集日志：通过参数配置收集GC日志；通过JDK工具查看GC状态。
4. 分析日志：使用工具辅助分析日志，查看GC次数，GC时间。
5. 调整参数：切换垃圾收集器或者调整垃圾收集器参数。

通用GC参数

-XX:ParallelGCThreads	并行GC线程数量
-XX:ConcGCThreads	并发GC线程数量
-XX:MaxGCPauseMillis	最大停顿时间，单位毫秒 GC尽力保证回收时间不超过设定值
-XX:GCTimeRatio	0-100的取值范围 垃圾收集时间占总时间的比 默认99，即最大允许1%时间做GC
-XX:SurvivorRatio	设置Eden区大小和Survivor区大小的比例 8表示 两个Survivor :Eden=2:8，即一个Survivor占年轻代的1/10
-XX:NewRatio	新生代和老年代的比 4 表示 新生代:老年代=1:4，即年轻代占堆的1/5
-verbose:gc 、 -XX:+printGC	打印GC的简要信息
-XX:+PrintGCDetails	打印GC详细信息
-XX:+PrintGCTimeStamps	打印CG发生的时间戳
-Xloggc:log/gc.log	指定GC log的位置，以文件输出
-XX:+PrintHeapAtGC	每次一次GC后，都打印堆信息

垃圾收集器Parallel参数调优

1. JDK默认的收集器
2. 吞吐量优先

-XX:+UseParallelGC	新生代使用并行回收收集器
-XX:+UseParallelOldGC	老年代使用并行回收收集器
-XX:ParallelGCThreads	设置用于垃圾回收的线程数
-XX:+UseAdaptiveSizePolicy	打开自适应 GC 策略

垃圾收集器CMS参数调优

1. 响应时间优先
2. Parallel GC无法满足应用程序延迟要求时再考虑使用CMS垃圾收集器。
3. 新版建议用G1垃圾收集器

-XX:+UseConcMarkSweepGC	新生代使用并行收集器，老年代使用CMS+串行收集器
-XX:+UseParNewGC	在新生代使用并行收集器,CMS下默认开启
-XX:CMSInitiatingOccupancyFraction	设置触发GC的阈值 默认68% 如果不幸内存预留空间不够，就会引起concurrent mode failure
-XX:+ UseCMSCompactAtFullCollection	Full GC后，进行一次整理 整理过程是独占的，会引起停顿时间变长
-XX:+CMSFullGCsBeforeCompaction	设置进行几次Full GC后，进行一次碎片整理
-XX:+CMSClassUnloadingEnabled	允许对类元数据进行回收
-XX:+UseCMSInitiatingOccupancyOnly	表示只在到达阀值的时候，才进行CMS回收
XX:+CMSIncrementalMode	使用增量模式，比较适合单 CPU

垃圾收集器G1参数调优

1. 兼顾吞吐量和响应时间
2. 超过50%的Java堆被实时数据占用。
3. 建议大堆（大小约为6GB或更大）
4. 且GC延迟要求有限的应用（稳定且可预测的暂停时间低于0.5秒）。

-XX:G1HeapRegionSize=<N, 例如 16>M	设置 region 大小，默认heap/2000
–XX:G1MixedGCLiveThresholdPercent	老年代依靠Mixed GC，触发阈值
–XX:G1OldCSetRegionThresholdPercent	定多被包含在一次 Mixed GC 中的 region 比例
-XX:+ClassUnloadingWithConcurrentMark	G1 增加并默认开启，在并发标记阶段结束后，JVM 即进行类型卸载。
-XX:G1NewSizePercent	新生代 的最小比例
-XX:G1MaxNewSizePercent	新生代的最大比例
-XX:G1MixedGCCountTarget	Mixed GC 数量控制

运行时JIT编译器优化参数

JIT编译值得是将字节码编译为本地代码执行，只有热点代码才会编译为本地代码。解释器执行节约内存，反之可以使用编译执行来提升效率。

-XX:+AggressiveOpts	允许jvm使用积极的性能优化功能
-XX:-TieredCompilation	分层编译 jdk8默认开启，jdk7默认关闭 client
-Xmaxjitcodesize、-XX:ReservedCodeCacheSize	指定JIT编译代码的最大代码高速缓存最大值
-Xmixed	除了热方法之外，解释器执行所有字节码，热方法被编译为本机代码
-XX:InitialCodeCacheSize	初始化编译后的汇编指令，缓存区大小，字节
-XX:+PrintCompilation	打开编译日志 jstat -compiler pid
-XX:CICompilerCount	JIT编译所使用的线程数量
-XX:+DoEscapeAnalysis	逃逸分析，默认打开。对代码的深度优化
-XX:-Inline	方法内联，默认打开。