**JVM的垃圾回收机制详解和调优**

gc即垃圾收集机制是指jvm用于释放那些不再使用的对象所占用的内存。java语言并不要求jvm有gc，也没有规定gc如何工作。不过常用的jvm都有gc，而且大多数gc都使用类似的算法管理内存和执行收集操作。

1.JVM的gc概述

gc即垃圾收集机制是指jvm用于释放那些不再使用的对象所占用的内存。java语言并不要求jvm有gc，也没有规定gc如何工作。不过常用的jvm都有gc，而且大多数gc都使用类似的算法管理内存和执行收集操作。

在充分理解了垃圾收集算法和执行过程后，才能有效的优化它的性能。有些垃圾收集专用于特殊的应用程序。比如，实时应用程序主要是为了避免垃圾收集中断，而大多数OLTP应用程序则注重整体效率。理解了应用程序的工作负荷和jvm支持的垃圾收集算法，便可以进行优化配置垃圾收集器。

垃圾收集的目的在于清除不再使用的对象。gc通过确定对象是否被活动对象引用来确定是否收集该对象。gc首先要判断该对象是否是时候可以收集。两种常用的方法是引用计数和对象引用遍历。

1.1.引用计数

引用计数存储对特定对象的所有引用数，也就是说，当应用程序创建引用以及引用超出范围时，jvm必须适当增减引用数。当某对象的引用数为0时，便可以进行垃圾收集。

1.2.对象引用遍历

早期的jvm使用引用计数，现在大多数jvm采用对象引用遍历。对象引用遍历从一组对象开始，沿着整个对象图上的每条链接，递归确定可到达（reachable）的对象。如果某对象不能从这些根对象的一个（至少一个）到达，则将它作为垃圾收集。在对象遍历阶段，gc必须记住哪些对象可以到达，以便删除不可到达的对象，这称为标记（marking）对象。

下一步，gc要删除不可到达的对象。删除时，有些gc只是简单的扫描堆栈，删除未标记的未标记的对象，并释放它们的内存以生成新的对象，这叫做清除（sweeping）。这种方法的问题在于内存会分成好多小段，而它们不足以用于新的对象，但是组合起来却很大。因此，许多gc可以重新组织内存中的对象，并进行压缩（compact），形成可利用的空间。

为此，gc需要停止其他的活动活动。这种方法意味着所有与应用程序相关的工作停止，只有gc运行。结果，在响应期间增减了许多混杂请求。另外，更复杂的gc不断增加或同时运行以减少或者清除应用程序的中断。有的gc使用单线程完成这项工作，有的则采用多线程以增加效率。

2.几种垃圾回收机制

2.1.标记－清除收集器

这种收集器首先遍历对象图并标记可到达的对象，然后扫描堆栈以寻找未标记对象并释放它们的内存。这种收集器一般使用单线程工作并停止其他操作。

2.2.标记－压缩收集器

有时也叫标记－清除－压缩收集器，与标记－清除收集器有相同的标记阶段。在第二阶段，则把标记对象复制到堆栈的新域中以便压缩堆栈。这种收集器也停止其他操作。

2.3.复制收集器

这种收集器将堆栈分为两个域，常称为半空间。每次仅使用一半的空间，jvm生成的新对象则放在另一半空间中。gc运行时，它把可到达对象复制到另一半空间，从而压缩了堆栈。这种方法适用于短生存期的对象，持续复制长生存期的对象则导致效率降低。

2.4.增量收集器

增量收集器把堆栈分为多个域，每次仅从一个域收集垃圾。这会造成较小的应用程序中断。

2.5.分代收集器

这种收集器把堆栈分为两个或多个域，用以存放不同寿命的对象。jvm生成的新对象一般放在其中的某个域中。过一段时间，继续存在的对象将获得使用期并转入更长寿命的域中。分代收集器对不同的域使用不同的算法以优化性能。

2.6.并发收集器

并发收集器与应用程序同时运行。这些收集器在某点上（比如压缩时）一般都不得不停止其他操作以完成特定的任务，但是因为其他应用程序可进行其他的后台操作，所以中断其他处理的实际时间大大降低。

2.7.并行收集器

并行收集器使用某种传统的算法并使用多线程并行的执行它们的工作。在多cpu机器上使用多线程技术可以显著的提高java应用程序的可扩展性。

3.Sun HotSpot 1.4.1 JVM堆大小的调整

Sun HotSpot 1.4.1使用分代收集器，它把堆分为三个主要的域：新域、旧域以及永久域。Jvm生成的所有新对象放在新域中。一旦对象经历了一定数量的垃圾收集循环后，便获得使用期并进入旧域。在永久域中jvm则存储class和method对象。就配置而言，永久域是一个独立域并且不认为是堆的一部分。

下面介绍如何控制这些域的大小。可使用-Xms和-Xmx 控制整个堆的原始大小或最大值。

下面的命令是把初始大小设置为128M：

java –Xms128m

–Xmx256m为控制新域的大小，可使用-XX：NewRatio设置新域在堆中所占的比例。

下面的命令把整个堆设置成128m，新域比率设置成3，即新域与旧域比例为1：3，新域为堆的1/4或32M：

java –Xms128m –Xmx128m–XX：NewRatio =3可使用-XX：NewSize和-XX：MaxNewsize设置新域的初始值和最大值。

下面的命令把新域的初始值和最大值设置成64m：

java –Xms256m –Xmx256m –Xmn64m

永久域默认大小为4m.运行程序时，jvm会调整永久域的大小以满足需要。每次调整时，jvm会对堆进行一次完全的垃圾收集。

使用-XX：MaxPerSize标志来增加永久域搭大小。在WebLogic Server应用程序加载较多类时，经常需要增加永久域的最大值。当jvm加载类时，永久域中的对象急剧增加，从而使jvm不断调整永久域大小。为了避免调整，可使用-XX：PerSize标志设置初始值。

下面把永久域初始值设置成32m，最大值设置成64m.

java -Xms512m -Xmx512m -Xmn128m -XX：PermSize=32m -XX：MaxPermSize=64m

默认状态下，HotSpot在新域中使用复制收集器。该域一般分为三个部分。第一部分为Eden，用于生成新的对象。另两部分称为救助空间，当Eden充满时，收集器停止应用程序，把所有可到达对象复制到当前的from救助空间，一旦当前的from救助空间充满，收集器则把可到达对象复制到当前的to救助空间。From和to救助空间互换角色。维持活动的对象将在救助空间不断复制，直到它们获得使用期并转入旧域。使用-XX：SurvivorRatio可控制新域子空间的大小。

同NewRation一样，SurvivorRation规定某救助域与Eden空间的比值。比如，以下命令把新域设置成64m，Eden占32m，每个救助域各占16m：

java -Xms256m -Xmx256m -Xmn64m -XX：SurvivorRation =2

如前所述，默认状态下HotSpot对新域使用复制收集器，对旧域使用标记－清除－压缩收集器。在新域中使用复制收集器有很多意义，因为应用程序生成的大部分对象是短寿命的。理想状态下，所有过渡对象在移出Eden空间时将被收集。如果能够这样的话，并且移出Eden空间的对象是长寿命的，那么理论上可以立即把它们移进旧域，避免在救助空间反复复制。但是，应用程序不能适合这种理想状态，因为它们有一小部分中长寿命的对象。最好是保持这些中长寿命的对象并放在新域中，因为复制小部分的对象总比压缩旧域廉价。为控制新域中对象的复制，可用-XX：TargetSurvivorRatio控制救助空间的比例（该值是设置救助空间的使用比例。如救助空间位1M，该值50表示可用500K）。该值是一个百分比，默认值是50.当较大的堆栈使用较低的sruvivorratio时，应增加该值到80至90，以更好利用救助空间。用-XX：maxtenuring threshold可控制上限。

为放置所有的复制全部发生以及希望对象从eden扩展到旧域，可以把MaxTenuring Threshold设置成0.设置完成后，实际上就不再使用救助空间了，因此应把SurvivorRatio设成最大值以最大化Eden空间，设置如下：

java … -XX：MaxTenuringThreshold=0 –XX：SurvivorRatio＝50000 …

4.BEA JRockit JVM的使用

Bea WebLogic 8.1使用的新的JVM用于Intel平台。在Bea安装完毕的目录下可以看到有一个类似于jrockit81sp1\_141\_03的文件夹。这就是Bea新JVM所在目录。不同于HotSpot把Java字节码编译成本地码，它预先编译成类。JRockit还提供了更细致的功能用以观察JVM的运行状态，主要是独立的GUI控制台（只能适用于使用Jrockit才能使用jrockit81sp1\_141\_03自带的console监控一些cpu及memory参数）或者WebLogic Server控制台。

Bea JRockit JVM支持4种垃圾收集器：

4.1.1.分代复制收集器

它与默认的分代收集器工作策略类似。对象在新域中分配，即JRockit文档中的nursery.这种收集器最适合单cpu机上小型堆操作。

4.1.2.单空间并发收集器

该收集器使用完整堆，并与背景线程共同工作。尽管这种收集器可以消除中断，但是收集器需花费较长的时间寻找死对象，而且处理应用程序时收集器经常运行。如果处理器不能应付应用程序产生的垃圾，它会中断应用程序并关闭收集。

分代并发收集器 这种收集器在护理域使用排它复制收集器，在旧域中则使用并发收集器。由于它比单空间共同发生收集器中断频繁，因此它需要较少的内存，应用程序的运行效率也较高，注意，过小的护理域可以导致大量的临时对象被扩展到旧域中。这会造成收集器超负荷运作，甚至采用排它性工作方式完成收集。

4.1.3.并行收集器

该收集器也停止其他进程的工作，但使用多线程以加速收集进程。尽管它比其他的收集器易于引起长时间的中断，但一般能更好的利用内存，程序效率也较高。

默认状态下，JRockit使用分代并发收集器。要改变收集器，可使用-Xgc：<gc\_name>，对应四个收集器分别为gencopy，singlecon，gencon以及parallel.可使用-Xms和-Xmx设置堆的初始大小和最大值。要设置护理域，则使用-Xns：java –jrockit –Xms512m –Xmx512m –Xgc：gencon –Xns128m…尽管JRockit支持-verbose：gc开关，但它输出的信息会因收集器的不同而异。JRockit还支持memory、load和codegen的输出。

注意 ：如果 使用JRockit JVM的话还可以使用WLS自带的console（C：\bea\jrockit81sp1\_141\_03\bin下）来监控一些数据，如cpu，memery等。要想能构监控必须在启动服务时startWeblogic.cmd中加入－Xmanagement参数。

5.如何从JVM中获取信息来进行调整

-verbose.gc开关可显示gc的操作内容。打开它，可以显示最忙和最空闲收集行为发生的时间、收集前后的内存大小、收集需要的时间等。打开-xx：+ printgcdetails开关，可以详细了解gc中的变化。打开-XX： + PrintGCTimeStamps开关，可以了解这些垃圾收集发生的时间，自jvm启动以后以秒计量。最后，通过-xx： + PrintHeapAtGC开关了解堆的更详细的信息。为了了解新域的情况，可以通过-XX：=PrintTenuringDistribution开关了解获得使用期的对象权。

6.Pdm系统JVM调整

6.1.服务器：前提内存1G 单CPU

可通过如下参数进行调整：－server 启用服务器模式（如果CPU多，服务器机建议使用此项）

－Xms，－Xmx一般设为同样大小。 800m

－Xmn 是将NewSize与MaxNewSize设为一致。320m

－XX：PerSize 64m

－XX：NewSize 320m 此值设大可调大新对象区，减少Full GC次数

－XX：MaxNewSize 320m

－XX：NewRato NewSize设了可不设。4

－XX： SurvivorRatio 4

－XX：userParNewGC 可用来设置并行收集

－XX：ParallelGCThreads 可用来增加并行度 4

－XXUseParallelGC 设置后可以使用并行清除收集器

－XX：UseAdaptiveSizePolicy 与上面一个联合使用效果更好，利用它可以自动优化新域大小以及救助空间比值

6.2.客户机：通过在JNLP文件中设置参数来调整客户端JVM

JNLP中参数：initial-heap-size和max-heap-size

这可以在framework的RequestManager中生成JNLP文件时加入上述参数，但是这些值是要求根据客户机的硬件状态变化的（如客户机的内存大小等）。建议这两个参数值设为客户机可用内存的60％（有待测试）。为了在动态生成JNLP时以上两个参数值能够随客户机不同而不同，可靠虑获得客户机系统信息并将这些嵌到首页index.jsp中作为连接请求的参数。

在设置了上述参数后可以通过Visualgc 来观察垃圾回收的一些参数状态，再做相应的调整来改善性能。一般的标准是减少fullgc的次数，最好硬件支持使用并行垃圾回收（要求多CPU）。