[译]Java8官方GC调优指南 --(六)平行收集器 - 掘金

juejin.cn/post/6844904053541961736

2020年1月28日

> 本套文章是Java8官方GC调优指南的全文翻译，点击查看原文,原文章名称《Java Platform, Standard Edition HotSpot Virtual Machine Garbage Collection Tuning Guide》

6 The Parallel Collector 平行收集器

The parallel collecor(AKA throughput collector)是一个分代收集器，与serial collector类似；唯一的区别是使用多线程提升gc速度。可以使用参数: `-XX:+UseParallelGC` 来启用parallel collector。默认情况，minor 和major gc都会使用parallel collector来做gc。

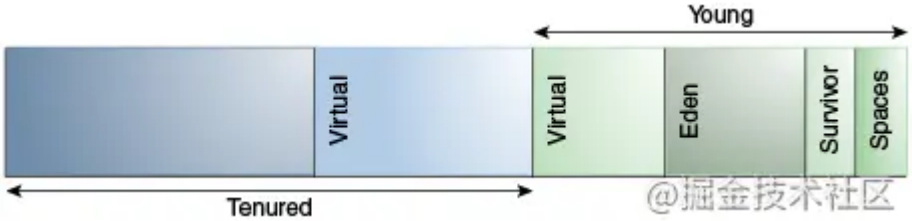
一台机器有N个物理线程并且N大于9时，平行收集器会使用一个固定的比率来选择gc线程数。这个分数约等于5/8.如果N小于8,那么gc线程数就是N。在一些特定的平台上，这个分数会降低到5/16.这个gc线程数可以通过参数来调整，具体参数后面再统一介绍。如果在一个单处理器的平台上，平行收集器不会像serial collector一样工作，因为多线程调度问题会有一些性能开销。不过，应用程序运行在中大型heap上时，它通常都要比双核机器上的serial collector 性能要好，在多核的机器上要比serial collector性能好很多。

gc线程数可以通过参数 `-XX:ParallelGCThreads=<N>` 来配置。如果使用参数对堆大小进行显式调优，则并行收集器所需的堆大小与串行收集器所需的堆大小相同。不过，开启parallel collector会让停顿时间变得更小。因为多个GC并发执行minor gc，一些碎片可能会从young区晋升到tenured区。每个gc线程持有一部分tenured区去做升代操作，所以可能产生一些碎片影响。减少gc线程数和增加tenured区大小可以减少这种影响。

> 对于fragmentation effect，个人理解就是多个线程并发gc时，每个线程负责收集young区的一部分，同时也可以将对象晋升到tenured区的部分区域。这样可能会让一些本来不应该晋升的对象晋升到了tenured区。细节还是要看源码，我后面如果了解了细节会做一些补充。

Generations 分代

之前提到过，parallel collector的分代策略是不同的。如下图所示：



Parallel Collector Ergonomics 平行收集器工效学

平行收集器是默认的服务器级收集器。还有，平行收集器使用自动调谐方法来允许你指定一些特定行为来替代分代大小和其他的底层调谐细节。你可以指定最大gc停顿时间，吞吐量，或者内存占用。

- Maximum Garbage Colletion Pause Time(最大GC停顿时间):通过参数 `-XX:MaxGCPauseMillis=<N>` 来指定。这个参数告诉JVM希望停顿时间不要超过这个值；默认情况下，没有这个限制。如果指定了这个参数，heap size和一些其他的相关参数会动态调整来降低停顿时间。这些调整可能会降低整个应用的吞吐量。
- Throughput(吞吐量):参数是 `-XX:GCTimeRatio=<N>` ，表示gc时间与应用时间的比例=1/(1+)。默认值是99，跟串行的差不多，不多说了。
- Footprint(内存占用):使用参数 `-Xmx<N>` 。收集器有一个含蓄的目标去降低heap大小，同时保证其他的目标能够达成。

Priority of Goals 目标优先级

这些目标会按照以下顺序处理：

1. Maximum pause time goal 最大停顿时间
2. Throughput 吞吐量
3. Minimum footprint goal 最小内存占用

最大停顿时间最优先保证。只有它达到预期后才会去处理吞吐量。只有前两个目标都达成之后才会考虑去做内存占用目标。

Generation Size Ajustments 调整分代内存大小

每次GC之后平均停顿时间都会被更新。检查目标是否满足，并且对分代大小做出调整。代码中的显示GC的停顿时间不会被计算到平均停顿时间。

分代大小的增加有一个固定的步长，是一个固定的百分比。分代内存的增加和降低操作是按照不同的频率来进行的。默认分代内存每次增加都会增加20%，减少则只会减少5%。可以通过参数来设置Young区每次增长的百分比 `-XX:YoungGenerationSizeIncrement=<Y>` 。Tenured区增长百分比通过参数 `--XTenuredGenerationSizeIncrement=<T>` 来控制。每个分代的减少的百分比通过一个系数参数来配置， `--XX:AdaptiveSizeDrementScaleFactor=<D>` 。如果young区增长百分比是X，那么young区每次减少的值是X/D%。

在GC时，如果收集器决定增加分代大小，会为这次增加的操作，再追加一些内存空间(叫做追加百分比)，也就是说，本来收集器要为young区加20%的内存，但是为了优化后续的操作，会追加25%，这多的5%就是追加百分比。这个追加量会随着收集次数增多而慢慢减少。追加这块内存是为了提升性能。减少分代大小时没有追加量。

如果最大停顿时间目标还没有达成，那么每次只会收缩一个分代大小。如果两个分代都达成了停顿时间目标，那么停顿时间长的那个分代会被优先收缩。

如果吞吐量目标还没有达成，两个分代都会增加。每一个都按其对总垃圾收集时间的占比来增加。例如，垃圾回收时间中25%是young gc，一次young区的分代内存增长比例是20%，那么依据这占比，young区大小会增加5%。

Default Heap Size 默认堆大小

除非初始化和最大heap size依据被指定了，否则heap size是被计算出来的。

Client JVM Default Initial and Maximum Heap Sizes 客户端JVM初始化和最大堆大小

默认情况下，如果物理机内存小于192MB，那么最大堆大小是物理机内存的一半，直到内存达到192MB。 或者是物理机内存1GB的4分之1。

例如，如果的电脑有128MB的内存，那么最大堆就是64MB，如果物理机内存大于或等于1GB那么堆内存等于256MB。 堆大小并不都是jvm使用的，除非你的程序创建了足够多的对象。 heap中3分之1是young区。

> 我们做Java后端的，这块儿就可以忽略了

Server JVM Default Initial and Maximum Heap Sizes 服务器JVM初始化和最大堆大小

和Client差不多。不过默认值会更高。在32bit JVM，默认的最大堆可以在4GB的物理内存下可以达到1GB。在64bit的JVM，默认堆内存可以到32GB，如果物理机有128GB以上的内存的话。(这个数差不多，笔者在一台512内存的docker宿主机上启动Gitlab容器，从宿主机上看到它占了32GB的内存)。下一节可以对这些值进行配置。

Specifying Initial and Maximum Heap Sizes 指定初始化和最大堆大小

通过 `-Xms` 和 `-Xmx` 参数来指定初始化和最大堆大小。如果你确定知道你程序具体能用多少heap，你可以将这两个值设置成一样的。如果你不清楚(我感觉除了跑批程序应该都不确定)，JVM会使用Xms来初始化heap，然后慢慢增长。

其他的参数和选项可以影响这些默认值。去确认你的默认值，使用 `-XX:+PrintFlagsFinal` 选项，在输出中查找MaxHeapSize。例如，在linux上，你可以执行下面的命令：

```
java -XX:+PrintFlagsFinal <GC options> -version | grep MaxHeapSize
```

复制代码

Excessive GC Time and OutOfMemoryError 过度GC时间和OOM

如果过多的时间花费在GC上，平行收集器会抛出OutOfMemoryError错误：如果98%以上的时间都花费在GC上，而且只能收集到2%的堆空间，OOM错误就会被抛出。这个特性是为了阻止程序运行在heap太小的情况下，当然了，内存泄漏也会导致OOM。如果有必要的话，可以禁用这个特性，配置这个参数即可: `-XX-UseGCOverheadLimit` 。

Measurements 度量

平行收集器的日志输出和serial collector的日志输出是一样的，不再赘述。