**Java垃圾回收器总结**

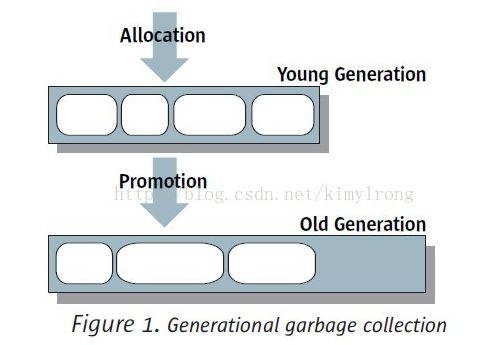
* 26679

什么是Java垃圾回收器

Java垃圾回收器是Java虚拟机(JVM)的三个重要模块(另外两个是解释器和多线程机制)之一，为应用程序提供内存的**自动分配(Memory Allocation)、自动回收(Garbage Collect)**功能，这两个操作都发生在Java堆上(一段内存快)。某一个时点，一个对象如果有一个以上的引用(Rreference)指向它，那么该对象就为活着的(Live)，否则死亡(Dead)，视为垃圾，可被垃圾回收器回收再利用。**垃圾回收操作需要消耗CPU、线程、时间等资源**，所以容易理解的是**垃圾回收操作不是实时的发生**(对象死亡马上释放)，当内存消耗完或者是达到某一个指标(Threshold,使用内存占总内存的比列，比如0.75)时，触发垃圾回收操作。有一个对象死亡的例外，java.lang.Thread类型的对象即使没有引用，只要线程还在运行，就不会被回收。

回收的机制

依据统计分析可知，Java(包括一些其它高级语言)里面**大多数对象生命周期都是短暂的**，所以把**Java内存分代管理**。分代的目的无非就是为不同代的内存块运用不同的管理策略(算法)，从而最大化性能。相对于年老代，通常年轻代要小很多，回收的频率高，速度快。年老代则回收频率低，耗时长。内存在年轻代里面分配，年轻代里面的对象经过多个回收周期依然存活的会自动晋升到年老代。



设计选型(Design Choices)

设计选型影响JVM垃圾回收器的实现难度，以及JVM的性能指标，适用于不同的场景。描述的是回收算法的风格特点。

单线程串行回收 VS 多线程并行回收

回收操作自身是否多线程处理的问题。单线程回收的优点是简单，易实现，碎片少，适用于单核的机器。多线程并行回收在多核机器上面可以充分的利用CPU资源，减少回收的时间，增加生产力，缺点是复杂且可能有部分碎片没有回收。

回收时暂停应用线程 VS 回收和应用并发进行

回收操作时是否暂停应用线程的问题。暂停应用线程的优点是简单、准确、清理得比较干净、清理的时间也短(CPU资源独占)，缺点是暂停应用线程之后会造成垃圾回收周期内应用的回应时间拉长，实时性非常高的系统比较敏感。回收和应用线程并行处理的优点是应用反应时间比较平稳、缺点是实现难度大、清理频率高、可能有碎片。

不合并释放的内存片段 VS 合并释放的内存片段 VS 把活着的复制到新的地方

这三个选型描述的是如何管理死亡的内存块片段。死亡的内存片段通常散落在堆的各个地方，如果不加以管理会有两个问题，内存分配的时候因查找可用的内存而导致速度慢，小的碎片会导致内存的浪费(比如大的数组要求大的连续内存片段)。管理有两种方式，把活着的内存挪到内存块的某一端，记录可用内存的开始位置，或者干脆把活着的内存复制到一个新的内存区域，原来的内存块整个空出来。

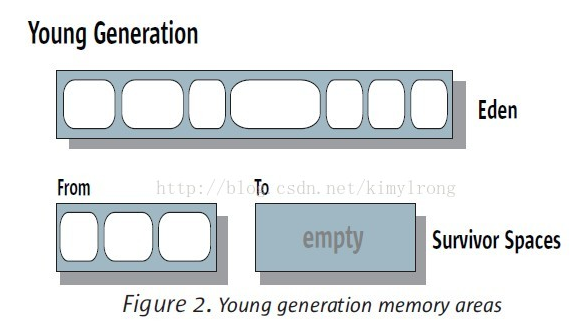
性能指标(Performance Metrics)

* **生产率(Throughput)**  
  一个较长的周期(长的周期才有意义)内，非回收时间占总时间的比率。度量系统的运行效率。
* **垃圾回收花费(Garbage Collection overhead)**  
  一个较长的周期内，回收时间占总时间的比率。与生产率相对应，加起来为100%。
* **暂停时间间隔(Pause time)**  
  Java虚拟机在回收垃圾的时候，有的算法会暂停所有应用线程的执行，某些系统可能对暂停的时间间隔比较敏感。
* **回收的频率(Frequency of collection)**  
  平均多久会发生回收操作。
* **内存占用的大小(Footprint)**  
  如堆的大小。
* **实时性(Promptness)**  
  自一个对象死亡起，经过多久该对象所占用内存被回收。

垃圾回收的类型

所有的回收器类型都是基于分代技术。Java HotSpot虚拟机包含三代，年轻代(Young Generation)、年老代(Old Generation)、永久代(Permanent Generation)。

* **永久代**  
  存储类、方法以及它们的描述信息。可以通过-XX:PermSize=64m和-XX:MaxPermSize=128m两个可选项指定初始大小和最大值。通常 我们不需要调节该参数，默认的永久代大小足够了，不过如果加载的类非常多，不够用了，调节最大值即可。
* **年老代**   
  主要存储年轻代中经过多个回收周期仍然存活从而升级的对象，当然对于一些大的内存分配，可能也直接分配到永久代(一个极端的例子是年轻代根本就存不下)。
* **年轻代**  
  绝大多数的内存分配回收动作都发生在年轻代。如下图所示, 年轻代被划分为三个区域，原始区(Eden)和两个小的存活区(Survivor)，两个存活区按功能分为From和To。绝大多数的对象都在原始区分配，超过一个垃圾回收操作仍然存活的对象放到存活区。



串行回收器(Serial Collector)

单线程执行回收操作，回收期间暂停所有应用线程的执行，client模式下的默认回收器，通过-XX:+UseSerialGC命令行可选项强制指定。

* **年轻代的回收算法(Minor Collection)**  
  把Eden区的存活对象移到To区，To区装不下直接移到年老代，把From区的移到To区，To区装不下直接移到年老代，From区里面年龄很大的升级到年老代。 回收结束之后，Eden和From区都为空，此时把From和To的功能互换，From变To，To变From，每一轮回收之前To都是空的。设计的选型为复制。
* **年老代的回收算法(Full Collection)**  
  年老代的回收分为三个步骤，标记(Mark)、清除(Sweep)、合并(Compact)。标记阶段把所有存活的对象标记出来，清除阶段释放所有死亡的对象，合并阶段 把所有活着的对象合并到年老代的前部分，把空闲的片段都留到后面。设计的选型为合并，减少内存的碎片。

并行回收器(Parallel Collector)

使用多个线程同时进行垃圾回收，多核环境里面可以充分的利用CPU资源，减少回收时间，增加JVM生产率，Server模式下的默认回收器。与串行回收器相同，**回收期间暂停所有应用线程的执行**。通过-XX:+UseParallelGC命令行可选项强制指定。

* **年轻代的回收算法(Minor Collection)**  
  使用多个线程回收垃圾，每一个线程的算法与串行回收器相同。
* **年老代的回收算法(Full Collection)**  
  年老代依然是单线程的，与串行回收器相同。

并行合并收集器(Parallel Compacting Collection)

年轻代和年老代的回收都是用多线程处理。通过命令可选项-XX:+UseParallelOldGC指定，–XX:ParallelGCThreads=3还可进一步指定参与并行回收的线程数。与串行回收器相同，**回收期间暂停所有应用线程的执行**。与并行回收器相比，年老代的回收时间更短，从而减少了暂停时间间隔(Pause time)。通过–XX:+UseParallelOldGC命令行可选项强制指定。

* **年轻代的回收算法(Minor Collection)**  
  与并行回收器(Parallel Collector)相同
* **年老代的回收算法(Full Collection)**   
  年老代分为三个步骤，标记、统计、合并。这里用到分的思想，把年老代划分为很多个固定大小的区(region)。 标记阶段，把所有存活的对象划分为N组(应该与回收线程数相同)，每一个线程独立的负责自己那一组，标记存活对象的位置以及 所在区(Region)的存活率信息，标记为并行的。统计阶段，统计每一个区(Region)的存活率，原则上靠前面的存活率较高，从前到后， 找到值得合并的开始位置(绝大多数对象都存活的区不值得合并)，统计阶段是串行的(单线程)。合并阶段，依据统计阶段的信息，多线程 并行的把存活的对象从一个区(Region)复制到另外一个区(Region)。

并发标记清除回收器(Concurrent Mark-Sweep Collector)

又名低延时收集器(Low-latency Collector)，通过各种手段使得应用程序被挂起的时间最短。**基本**与应用程序并发地执行回收操作，没有合并和复制操作。通过命令行-XX:+UseConcMarkSweepGC指定，在单核或者双核系统里面还可以指定使用增量式回收模式-XX:+UseConcMarkSweepGC。增量式回收是指把回收操作分为多个片段，执行一个片段之后释放CPU资源给应用程序，未来的某个时点接着上次的结果继续回收下去。目的也是减少延时。

* **年轻代的回收算法(Minor Collection)**  
  与并行回收器(Parallel Collector)相同
* **年老代的回收算法(Full Collection)**   
  分为四个步骤，初始标记(Initial Mark)、并发标记(Concurrent Mark)、再次标记(Remark)、以及并发清理(Concurrent Sweep)。特别注意，没有合并操作，所以会有碎片。
* 初始化阶段: 暂停应用线程，找出所有存活的对象，耗时比较短，回收器使用单线程。
* 并发标记阶段: 回收器标记操作与应用并发运行，回收器使用单线程标记存活对象。
* 再次标记：并发标记阶段由于应用程序也在运行，这个过程中可能新增或者修改对象。所以再次**暂停应用线程**，找出所有修改的对象，使用多线程标记。
* 并发清理：回收器清理操作与应用并发运行，回收器使用单线程清理死亡对象。

Java垃圾回收器的性能评估工具

* –XX:+PrintGCDetails和–XX:+PrintGCTimeStamps  
  垃圾回收的开始时间，持续时间，每一代的空余内存等信息。
* jmap [options] pid  
  jmap 2043 查看2043进程里面已经加载的共享对象。通常DLL文件。  
  jmap -heap 2043 查看内存堆的配置信息以及使用情况。  
  jmap -permstat 2043 查看永久代的加载情况。  
  jmap -histo 2043 查看类的加载和内存占用情况。
* jstat [options] pid  
  jstat -class 2043 class加载、卸载、内存占用情况。  
  jstat -gc 2043 GC执行情况。

后记

Java提供自动选择和自动性能优化功能。在做垃圾回收器调优之前，先列出所关注的性能指标，通过命令行告诉JVM你所关注的性能指标，由JVM自动调优，如果不满意，可以指定垃圾回收器。OutOfMemory通常是由于堆内存不足，调节-Xmx1024m和-XX:MaxPermSize=128m命令行可选项即可。

参考链接

[*Java Memory Management Whitepaper*](http://www.oracle.com/technetwork/java/javase/memorymanagement-whitepaper-150215.pdf?ssSourceSiteId=otncn)