对于Java开发人员来说，了解垃圾回收机制（GC）有哪些好处呢？首先可以满足作为一名软件工程师的求知欲，其次，深入了解GC如何工作可以帮你写出更好的Java应用。

这仅仅代表我个人的意见，但我坚信一个精通GC的人往往是一个好的Java开发者。如果你对GC的处理过程感兴趣，说明你已经具备较大规模应用的开发经验。如果你曾经想过如何正确的选择GC算法，那意味着你已经完全理解你所开发的应用的特点。当然，我们不能以偏概全，这不能作为评价一个好的开发人员的共通标准。但是，我要说的是，深入理解GC是成为一名伟大的程序员的必经之路。

这是成为JavaGC专家系列文章的第一篇，本篇主要针对GC机制进行介绍，在下一篇中，我们将重点探讨分析GC状态以及来自NHN的GC调优的例子。

本文的目的是以一种简单的方式向你介绍GC机制。我希望这些文章能够帮到你。实际上，我的学生已经在Twitter上发布了[一些很好的关于Java内核的文章](http://www.cubrid.org/blog/tags/Java/" \t "http://www.importnew.com/_blank)，并且大受欢迎。有兴趣的话，你也可以关注他们。

回到正题，咱们继续谈垃圾回收，在学习GC之前，你首先应该记住一个单词：“****stop-the-world****”。Stop-the-world会在任何一种GC算法中发生。Stop-the-world意味着 JVM 因为要执行GC而停止了应用程序的执行。当Stop-the-world发生时，除了GC所需的线程以外，所有线程都处于等待状态，直到GC任务完成。GC优化很多时候就是指减少Stop-the-world发生的时间。

****按代的垃圾回收机制****

在Java程序中不能显式地分配和注销内存。有些人把相关的对象设置为null或者调用System.gc()来试图显式地清理内存。设置为null至少没什么坏处，但是调用System.gc()会显著地影响系统性能，必须彻底杜绝（还好，我还没有见到NHN的哪个开发者调用这个方法）。

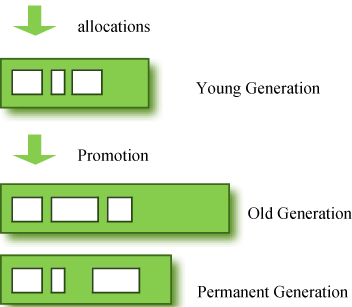
在Java中，开发人员无法直接在程序代码中清理内存，而是由垃圾回收器自动寻找不必要的垃圾对象，并且清理掉他们。垃圾回收器会在下面两种假设（hypotheses）成立的情况下被创建（称之为假设不如改为推测（suppositions）或者前提（preconditions））。

* 大多数对象会很快变得不可达
* 只有很少的由老对象（创建时间较长的对象）指向新生对象的引用

这些假设我们称之为****弱年代假设****（ ****weak generational hypothesis****）。为了强化这一假设，HotSpot虚拟机将其物理上划分为两个–新生代（****young generation****）和老年代（****old generation****）。  
****新生代（Young generation）:****绝大多数最新被创建的对象会被分配到这里，由于大部分对象在创建后会很快变得不可到达，所以很多对象被创建在新生代，然后消失。对象从这个区域消失的过程我们称之为”****minor GC****“。

****老年代（Old generation）:**** 对象没有变得不可达，并且从新生代中存活下来，会被拷贝到这里。其所占用的空间要比新生代多。也正由于其相对较大的空间，发生在老年代上的GC要比新生代少得多。对象从老年代中消失的过程，我们称之为”****major GC****“（或者”****full GC****“）

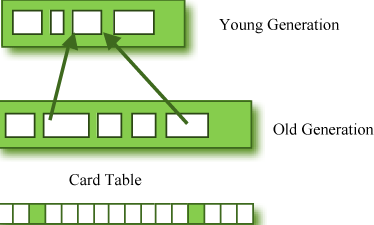
请看下面这个图表。

[](http://www.importnew.com/wp-content/uploads/2012/12/Figure-1-GC-Area-Data-Flow.png)

****图1 : GC 空间 & 数据流****

上图中的持久代（ ****permanent generation**** ）也被称为****方法区****（****method area****）。他用来保存类常量以及字符串常量。因此，这个区域不是用来永久的存储那些从老年代存活下来的对象。这个区域也可能发生GC。并且发生在这个区域上的GC事件也会被算为major GC。

有些人可能会问：  
****如果老年代的对象需要引用一个新生代的对象，会发生什么呢？****  
为了解决这个问题，老年代中存在一个”****card table****“，他是一个512 byte大小的块。所有老年代的对象指向新生代对象的引用都会被记录在这个表中。当针对新生代执行GC的时候，只需要查询card table来决定是否可以被收集，而不用查询整个老年代。这个card table由一个****write barrier****来管理。write barrier给GC带来了很大的性能提升，虽然由此可能带来一些开销，但GC的整体时间被显著的减少。

[](http://www.importnew.com/wp-content/uploads/2012/12/Figure2-Card-Table-Structure.png)

****图 2: Card Table 结构****

****新生代的构成****

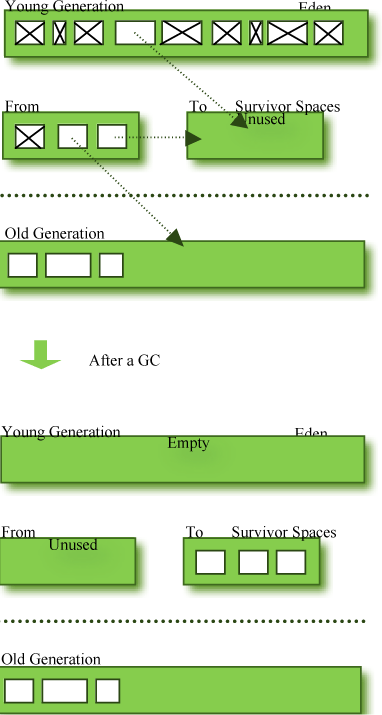
为了更好地理解GC，我们现在来学习新生代，新生代是用来保存那些第一次被创建的对象，他可以被分为三个空间

* 一个伊甸园空间（****Eden****）
* 两个幸存者空间（****Survivor**** ）

一共有三个空间，其中包含两个幸存者空间。每个空间的执行顺序如下：

1. 绝大多数刚刚被创建的对象会存放在伊甸园空间。
2. 在伊甸园空间执行了第一次GC之后，存活的对象被移动到其中一个幸存者空间。
3. 此后，在伊甸园空间执行GC之后，存活的对象会被堆积在同一个幸存者空间。
4. 当一个幸存者空间饱和，还在存活的对象会被移动到另一个幸存者空间。之后会清空已经饱和的那个幸存者空间。
5. 在以上的步骤中重复几次依然存活的对象，就会被移动到老年代。

如果你仔细观察这些步骤就会发现，其中一个幸存者空间必须保持是空的。如果两个幸存者空间都有数据，或者两个空间都是空的，那一定标志着你的系统出现了某种错误。  
通过频繁的minor GC将数据移动到老年代的过程可以用下图来描述：

[](http://www.importnew.com/wp-content/uploads/2012/12/Figure-3-Before-After-a-GC.png)

****图 3: GC执行前后对比****

需要注意的是HotSpot虚拟机使用了两种技术来加快内存分配。他们分别是是”****bump-the-pointer****“和“****TLABs（Thread-Local Allocation Buffers）”。****

****Bump-the-pointer****技术跟踪在伊甸园空间创建的最后一个对象。这个对象会被放在伊甸园空间的顶部。如果之后再需要创建对象，只需要检查伊甸园空间是否有足够的剩余空间。如果有足够的空间，对象就会被创建在伊甸园空间，并且被放置在顶部。这样以来，每次创建新的对象时，只需要检查最后被创建的对象。这将极大地加快内存分配速度。但是，如果我们在多线程的情况下，事情将截然不同。如果想要以线程安全的方式以多线程在伊甸园空间存储对象，不可避免的需要加锁，而这将极大地的影响性能。****TLABs**** 是HotSpot虚拟机针对这一问题的解决方案。该方案为每一个线程在伊甸园空间分配一块独享的空间，这样每个线程只访问他们自己的TLAB空间，再与bump-the-pointer技术结合可以在不加锁的情况下分配内存。  
以上是针对新生代空间GC技术的简要介绍，你不需要刻意记住我刚刚提到的两种技术。不知道他们不会对你产生什么影响，但是请务必记住在对象刚刚被创建之后，是保存在伊甸园空间的。那些长期存活的对象会经由幸存者空间转存在老年代空间。

****老年代GC处理机制****

老年代空间的GC事件基本上是在空间已满时发生，执行的过程根据GC类型不同而不同，因此，了解不同的GC类型将有助于你理解本节的内容。  
JDK7一共有5种GC类型：

1. Serial GC
2. Parallel GC
3. Parallel Old GC (Parallel Compacting GC)
4. Concurrent Mark & Sweep GC  (or “CMS”)
5. Garbage First (G1) GC

其中，Serial GC不应该被用在服务器上。这种GC类型在单核CPU的桌面电脑时代就存在了。使用Serial GC会显著的降低应用的性能指标。  
现在，让我们共同学习每一种GC类型

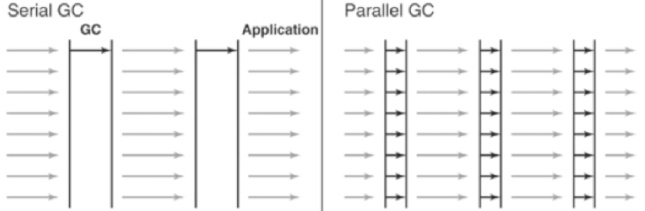
****1. Serial GC (-XX:+UseSerialGC)****

新生代空间的GC方式我们在前面已经介绍过了，在老年代空间中的GC采取称之为”****mark-sweep-compact****“的算法。

1. 算法的第一步是标记老年代中依然存活对象。（标记）
2. 第二步，从头开始检查堆内存空间，并且只留下依然幸存的对象。（清理）

最后一步，从头开始，顺序地填满堆内存空间，并且将对内存空间分成两部分：一个保存着对象，另一个空着（压缩）。

****2. Parallel GC (-XX:+UseParallelGC)****

[](http://www.importnew.com/wp-content/uploads/2012/12/Figure-4-Difference-between-the-Serial-GC-and-Parallel-GC.png)

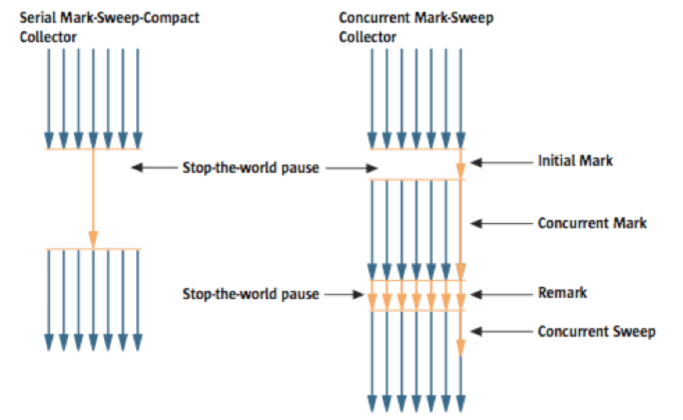
****图 4: Serial GC 与 Parallel GC的区别****

从上图中，你可以轻易地看出serial GC和parallel GC的区别，serial GC只使用一个线程执行GC，而parallel GC使用多个线程，因此parallel GC更高效。这种GC在内存充足以及多核的情况下会很有用，因此我们也称之为”****throughput GC****“。

****3. Parallel Old GC(-XX:+UseParallelOldGC)****

Parallel Old GC在JDK5之后出现。与parallel GC相比，唯一的区别在于针对老年代的GC算法。Parallel Old GC分为三步：标记-汇总-压缩（mark – summary – compaction）。汇总（summary）步骤与清理（sweep）的不同之处在于，其将依然幸存的对象分发到GC预先处理好的不同区域，算法相对清理来说略微复杂一点。

****4. CMS GC (-XX:+UseConcMarkSweepGC)****

[](http://www.importnew.com/wp-content/uploads/2012/12/Figure-5-Serial-GC-CMS-GC.png)

****图 5: Serial GC & CMS GC****

就像你从上图看到的那样, CMS GC比我之前解释的各种算法都要复杂很多。第一步初始化标记（initial mark） 比较简单。这一步骤只是查找那些距离类加载器最近的幸存对象。因此，停顿的时间非常短暂。在之后的并行标记（ concurrent mark ）步骤，所有被幸存对象引用的对象会被确认是否已经被追踪和校验。这一步的不同之处在于，在标记的过程中，其他的线程依然在执行。在重新标记（remark）步骤，会再次检查那些在并行标记步骤中增加或者删除的与幸存对象引用的对象。最后，在并行交换（ concurrent sweep ）步骤，转交垃圾回收过程处理。垃圾回收工作会在其他线程的执行过程中展开。一旦采取了这种GC类型，由GC导致的暂停时间会极其短暂。CMS GC也被称为低延迟GC。它经常被用在那些对于响应时间要求十分苛刻的应用之上。

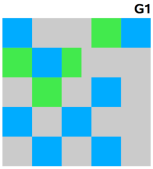
当然，这种GC类型在拥有stop-the-world时间很短的优点的同时，也有如下缺点：

* 它会比其他GC类型占用更多的内存和CPU
* 默认情况下不支持压缩步骤

在使用这个GC类型之前你需要慎重考虑。如果因为内存碎片过多而导致压缩任务不得不执行，那么stop-the-world的时间要比其他任何GC类型都长，你需要考虑压缩任务的发生频率以及执行时间。

****5. G1 GC****

最后，我们来学习垃圾回收优先（G1）GC类型。

[](http://www.importnew.com/wp-content/uploads/2012/12/Figure-6-Layout-of-G1-GC.png)

****图 6:  G1 GC的结构****

 如果你想要理解G1，首先你要忘记你所学过的新生代和老年代的概念。正如你在上图所看到的，每个对象被分配到不同的格子，随后GC执行。当一个区域装满之后，对象被分配到另一个区域，并执行GC。这中间不再有从新生代移动到老年代的三个步骤。这个类型是为了替代CMS GC而被创建的，因为CMS GC在长时间持续运作时会产生很多问题。

G1最大的好处是性能，他比我们在上面讨论过的任何一种GC都要快。但是在JDK 6中，他还只是一个早期试用版本。在JDK7之后才由官方正式发布。就我个人看来，NHN在将JDK 7正式投入商用之前需要很长的一段测试期（至少一年）。因此你可能需要再等一段时间。并且，我也听过几次使用了JDK 6中的G1而导致Java虚拟机宕机的事件。请耐心的等到它更稳定吧。

下一次我将讨论GC优化相关的问题，但是在此之前我要先明确一件事情，假如应用中创建的所有对象的大小和类型都是统一的，那么公司使用的WAS的GC参数可以是相同的。但是WAS所创建对象的大小和生命周期根据服务以及硬件的不同而不同。换句话说，不能因为某个应用使用的GC参数“A”，就说明同样的参数也能给其他服务带来最佳的效果。而是要因地制宜，有的放矢。我们需要找到适合每个WAS线程的参数，并且持续的监控和优化每个设备上的WAS实例。这并不是我的一家之谈，而是负责Oracle Java虚拟机研发的工程师在 JavaOne 2010上已经讨论过的。

本文中我们简略的介绍了Java的GC机制，请继续关于我们的后续文章，我们将会讨论****如何监控Java GC状态以及优化GC。****

另外，我特别推荐一本2011年12月发布的《Java性能》（[Amazon](http://amzn.com/0137142528" \t "http://www.importnew.com/_blank),也可以通过safari在线阅读），还有在Oracle官网发布的白皮书《Java HotSpotTM虚拟机内存管理》（这本书与Java性能优化不是同一本） 作者Sangmin Lee, NHN公司，性能工程师实验室高级工程师。