**1、 回收算法**

**1.1、Java堆对象回收判断**

（1）**引用计数法**

给对象添加一引用计数器，被引用一次计数器值就加 1，实现简单，但是无法解决循环依赖

（2）**可达性分析法（默认）**

从GC Roots 到这个对象是否可达，在Java语言中，可作为 GC Roots 的对象包括下面几种：

a. 虚拟机栈（栈帧中的局部变量表）中引用的对象。

b. 方法区中常量或类静态属性引用的对象。

c. 本地方法栈中 JNI（Native方法）引用的对象

**1.2、永久代（方法区）回收判断**

永久代的垃圾收集主要回收两部分内容：废弃常量和无用的类。

回收废弃常量与回收 Java 堆中的对象非常类似。以常量池中字面量的回收为例，假如一个字符串"abc"已经进入了常量池中，但是当前系统没有任何一个 String 对象是叫做"abc"的，也没有其他地方引用了这个字面量，如果这时发生内存回收，而且必要的话，这个"abc"常量就会被系统清理出常量池。常量池中的其他类（接口）、方法、字段的符号引用也与此类似。

类需要同时满足下面 3 个条件才能算是“无用的类”：

a. 该类所有的实例都已经被回收，也就是 Java 堆中不存在该类的任何实例。

b. 加载该类的 ClassLoader 已经被回收。

c. 该类对应的 java.lang.Class 对象没有在任何地方被引用，无法在任何地方通过反射访问该类的方法。

虚拟机可以对满足上述 3 个条件的无用类进行回收，这里说的仅仅是“可以”，而并不是和对象一样，不使用了就必然会回收。

**1.3、垃圾收集算法**

一共有 4 种

（1）**标记-清除算法**

标记和清楚效率不高，有内存碎片，适用于老年代

（2）**复制算法**

将内存分为2块，每次只使用1块

现在的商业虚拟机都采用这种算法来回收新生代，由于新生代中大部分是“朝生夕死”的，所以并不需要按照 1:1 的比例来划分2块内存空间，而是将内存划分为一块较大的 Eden 空间和两块较小的 Survivor 空间，每次使用 Eden 和其中一块 Survivor 。

当回收时，将 Eden 和 Survivor 中还存活着的对象一次性地复制到另外一块 Survivor 空间上，最后清理掉 Eden 和刚才用过的 Survivor 空间。HotSpot 虚拟机默认 Eden:Survivor = 8:1，也就是每次新生代中可用内存空间为整个新生代容量的 90%（其中一块Survivor不可用），只有 10% 的内存会被“浪费”。

当然，我们没有办法保证每次回收都只有不多于 10% 的对象存活，当 Survivor 空间不够用时，需要依赖其他内存（这里指老年代）进行分配担保。

（3）**标记-整理算法**

根据老年代的特点，有人提出了另外一种“标记-整理”算法，标记过程仍然与“标记-清除”算法一样，但后续步骤不是直接对可回收对象进行清理，而是让所有存活的对象都向一端移动，然后直接清理掉端边界以外的内存。

（4）**分代收集算法**

当前商业虚拟机的垃圾收集都采用“分代收集”算法，根据对象存活周期的不同将内存划分为几块并采用不用的垃圾收集算法。

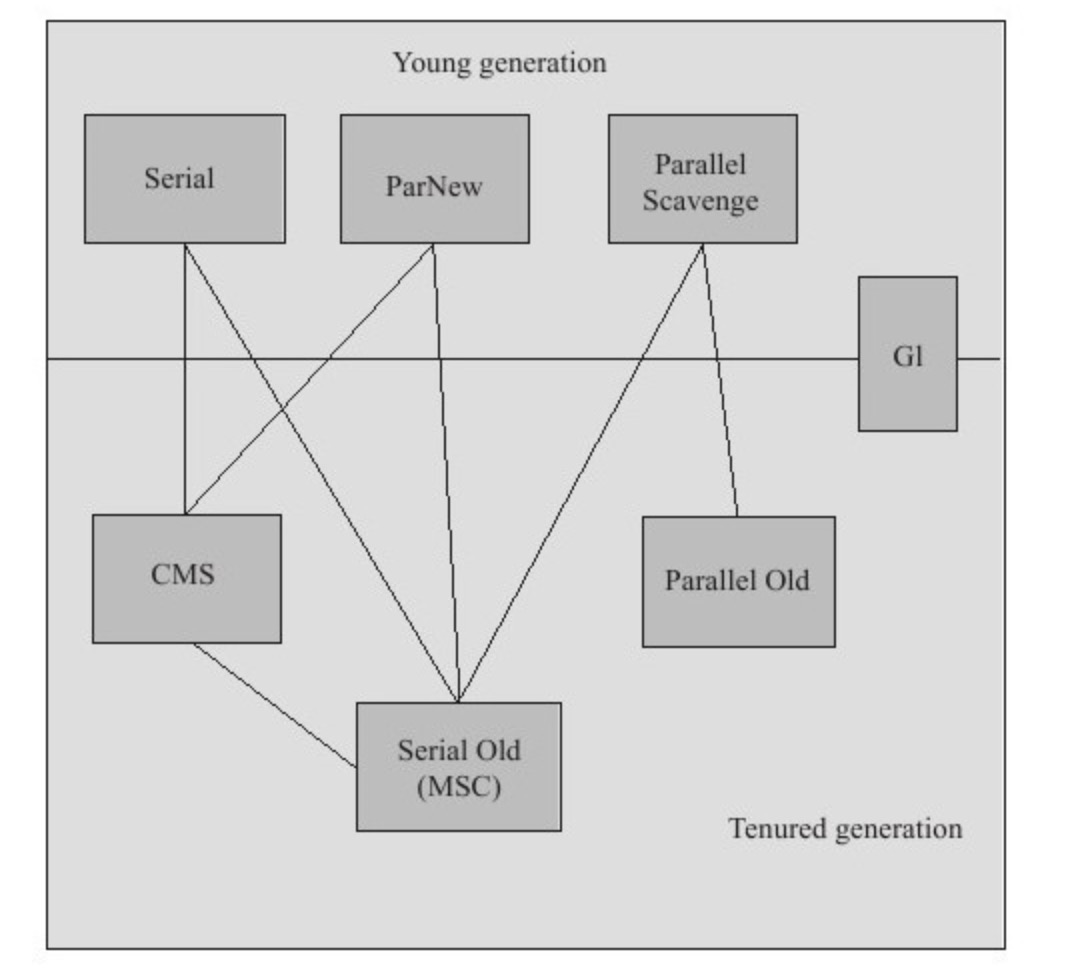
一般是把 Java 堆分为新生代和老年代，这样就可以根据各个年代的特点采用最适当的收集算法。在新生代中，选用复制算法，老年代中因为对象存活率高、没有额外空间对它进行分配担保，就必须使用“标记—清理”或者“标记—整理”算法来进行回收。

**1.4、垃圾收集器**

基于JDK 1.7 Update 14之后的 HotSpot 虚拟机里的垃圾收集器有如下7种，其中JDK7、8默认是parallel scavenge（并行收集器，相比parNew，更注重吞吐量）和parallel old（并行手收集器），JDK9是G1；CMS和G1是并发收集器，来看看并行和并发的区别

并行（Parallel）：指多条垃圾收集线程并行工作，但此时用户线程仍然处于等待状态。

并发（Concurrent）：指用户线程与垃圾收集线程同时执行（但不一定是并行的，可能会交替执行），用户程序在继续运行，而垃圾收集程序运行于另一个 CPU 上。



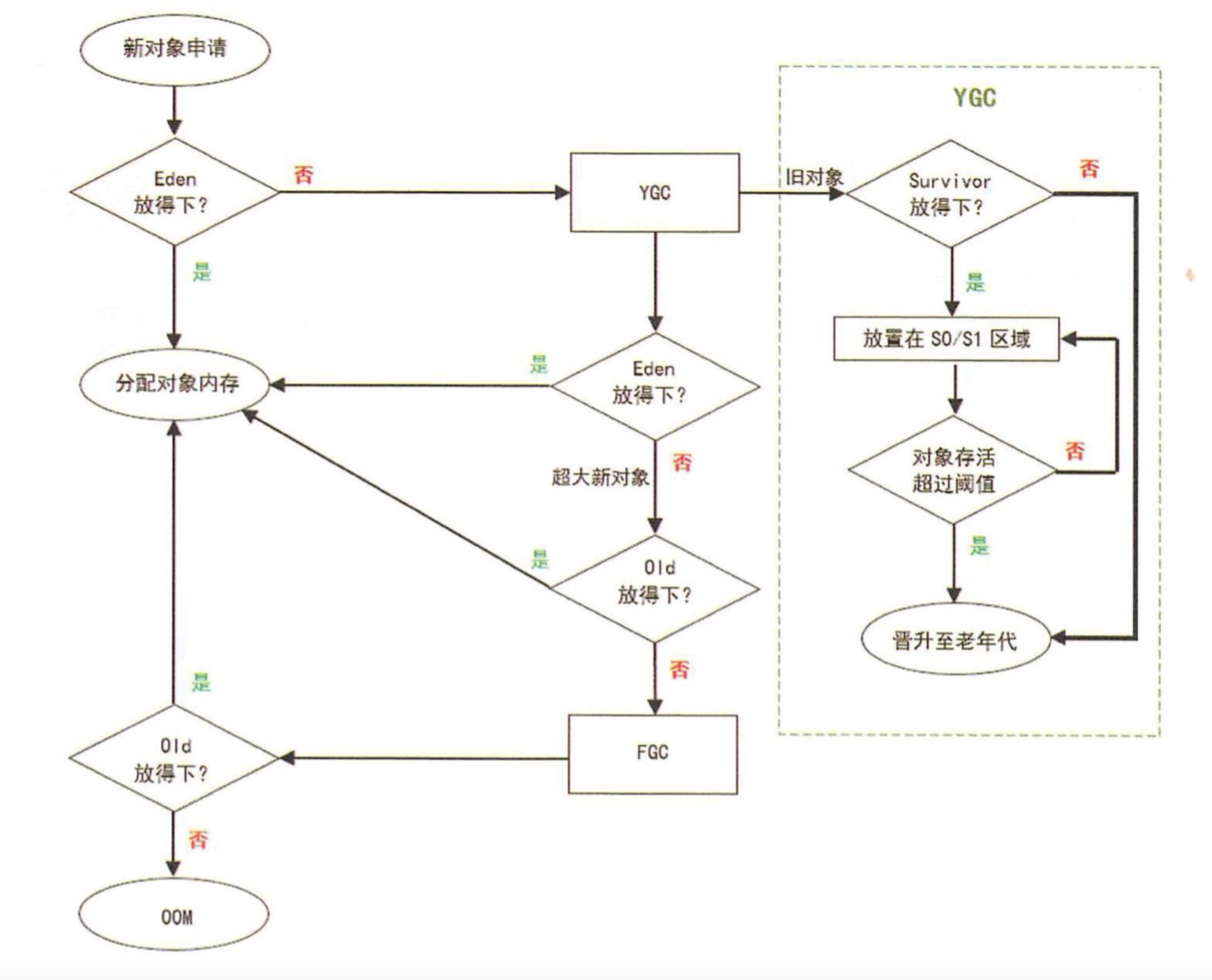
并行收集器需要stop the world，而并发收集器却可以用户、GC线程同时进行（其实有短暂的stop the world，不过可以忽略），是怎么做到的呢？核心就在于并发标记的实现

并发标记：一般采用三色标记法，黑色--根对象或者该对象与它子对象都被扫描；灰色--对象本身被扫描，但子对象未被扫描；白色--对象本身未被扫描

为了防止用户线程运行过程中丢失标记的对象，一般采用“插入时记录”或者“删除时记录”，同时会采用短暂的stop the world来重新标记或最终标记，然后一次性清除

**1.5、内存分配与回收策略**

对象的内存分配，往大方向讲，就是在堆上分配，堆又主要分为新生代和老年代（默认比例为1:2），新生代又分为一个Eden和两个Survivor，对象主要分配在Eden区上。少数情况下也可能会直接分配在老年代中，分配的规则并不是百分之百固定的，其细节取决于当前使用的是哪一种垃圾收集器组合，还有虚拟机中与内存相关的参数的设置。一般情况下，如果有新对象申请，Eden放不下就会触发young-gc，如果Young-GC（频繁、耗时短）后Eden和老年代都还是放不下，那么就会触发Full-GC（耗时比较长）



1、对象优先在Eden分配

2、大对象直接进入老年代（-XX:PretenureSizeThreshold，默认3M，Parallel收集器不生效）

3、长期存活的对象将进入老年代（-XX:MaxTenuringThreshold，默认15）

4、同龄对象达到 Survivor 空间的一半后，大于等于该年龄的对象都将进入老年代

5、空间分配担保

什么是空间分配担保？

在发生 Young-GC 之前，虚拟机会先检查老年代最大可用的连续空间够不够用，判断分为两步：

第一步判断（这个判断虚拟机会先检查老年代最大可用的连续空间是否大于新生代所有对象总空间，如果这个条件成立，那么Young-GC可以确保是安全的，可以直接执行），如果不成立，则还需要进行第二步的判断，毕竟第一步的判断范围有点大。

第二步判断，这次是个概率判断，虚拟机会查看HandlePromotionFailure（jdk7后参数不生效，默认允许）设置值是否允许担保失败。如果不允许，则执行Full -GC，如果允许，则继续检查老年代最大可用的连续空间是否大于历次晋升到老年代对象的平均大小，如果大于，将尝试着进行一次Young-GC（尽管是有风险的），如果小于Full GC。