1. 标记-清除

这个算法主要分为两部：标记和清除

首先标记出要被回收的对象、在标记完成后统一进行回收所有的对象

不足之处：

首先是效率问题，无论是标记还是清楚的效率都很低、其次是内存空间问题，会导致产生大量不连续的内存空间碎片，空间碎片会导致当分配大的对象的时候，没有足够连续的空间进而不得不提前触发gc动作

2. 复制算法

复制算法即将内存空间分配为大小相等的两份内存空间，每次只使用一块，当这一块使用完了，就将还存活的对象复制到另一个上，然后再把已经使用的内存空间进行一次清掉，这种算法每次只对半个区域进行垃圾回收、内存分配时就不用考虑内存碎片的问题等复杂情况、只要移动堆顶指针、按顺序分配即可，实现简单、效率高，但是至终算法的代价就是将内存缩小为原来的一半、代价有些高

目前商业虚拟机都采用这种方式来回收新生代，研究表明，新生代的对象98%朝生夕死，因此不能按照1：1分配，而是将内存分配一个较大的Eden空间和两块教小的Survivor空间，每次使用Eden和其中一块Survivor空间，当回收的时候，将Eden和Survior中还存活的对象一次性复制到另一块Survivor空间上，最后清空Eden和Survior空间。HotSpot虚拟机默认的Eden和Survior的大小比例为8:1。当survivor的空间不足的时候，需要依赖其他内存（老年代）进行分配担保

分配担保：如果另一块Survivor空间没有足够的空间存放上一次新生代收集下来的存活对象，这些对象将直接通过分配担保机制进入老年代

3. 标记-整理算法

复制算法在存活率教高的情况下进行复制操作，就会导致效率很低，所以老年代不能使用这种算法

根据老年代的特征：有人提出另外一种算法：标记-整理。标记过程与标记清除过程一样，但是后续不是直接对对象进行清理，而是让所有存活的对象向一端移动，然后直接清理掉另一端的内存

4. 分代收集算法

目前的虚拟机中使用的就是这种思路，根据对象存活周期将内存划分为寄快，一

般是把java堆分为新生代和老年代，然后根绝各个年代的特征采用合适的收集算法。在新生代，每次垃圾收集都会有大批对象死亡，因此使用复制算法，而老年代存活率高，没有额外的空间对她们进行分配担保，因此使用“标记-清理"或者”标记-整理“算法进行回收