垃圾回收 （Grabage Collection）

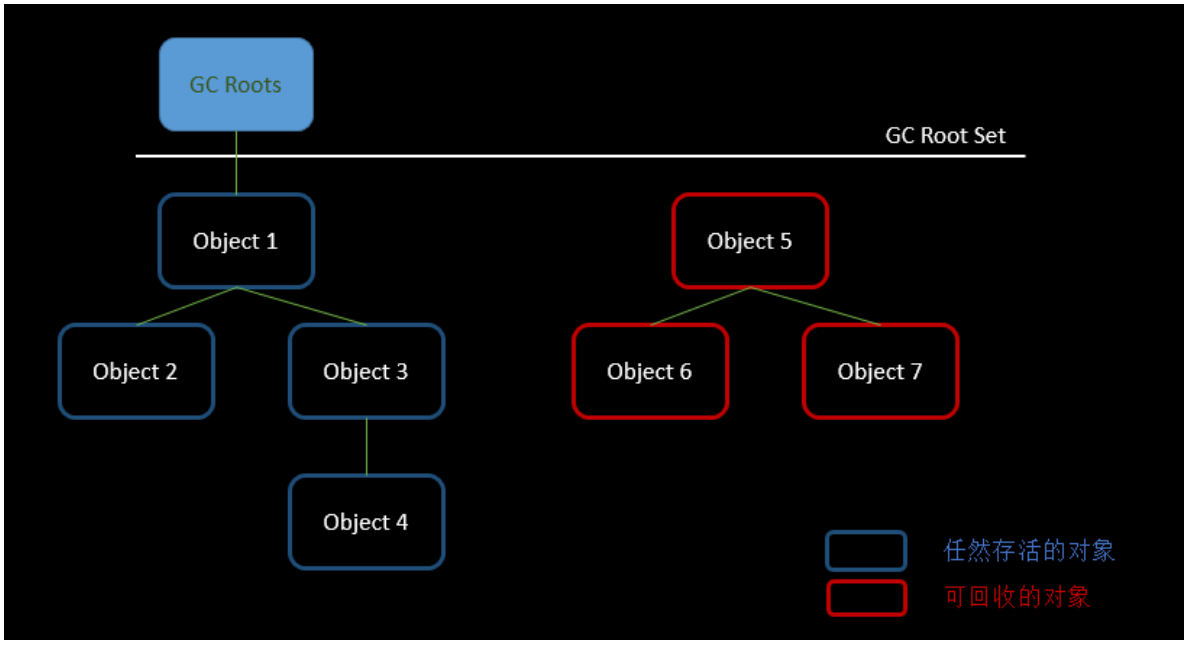
> 垃圾回收算法
> minor GC和major GC的区别
> 内存分代

为什么垃圾回收

因为Java是在JVM中运行的，所有托管资源（内存）由JVM自动回收，让开发人员更多关注于业务。

回收那些内存

在`JVM`中一般按照可达性分析算法来分析，查看那些对象（资源）应该被释放。一般没有被`GC Roots`引用的对象，就是可回收对象。
如图gc roots



根对象

编号	说明
No.1	虚拟机栈中的引用对象
No.2	方法区中的类静态属性引用的对象
No.3	方法区中的常量引用的对象
No.4	本地方法栈中`JNI`(`Native`方法)

如何回收

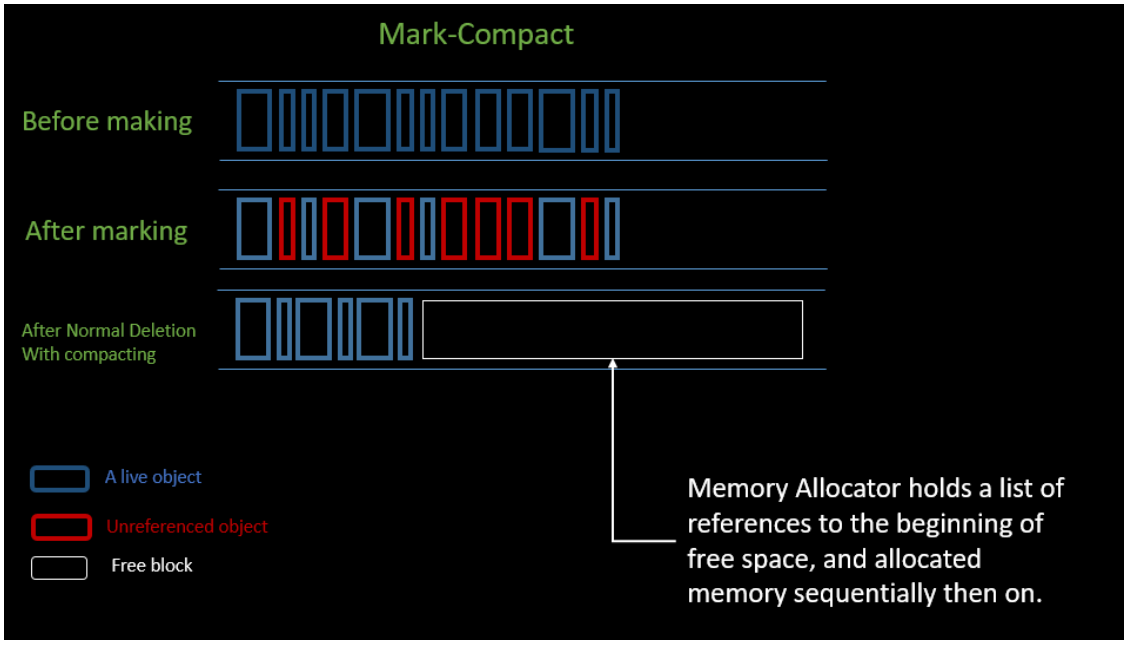
`JVM`提供了两种回收算法。

1. `Marking-Sweep`(标记-清除法)

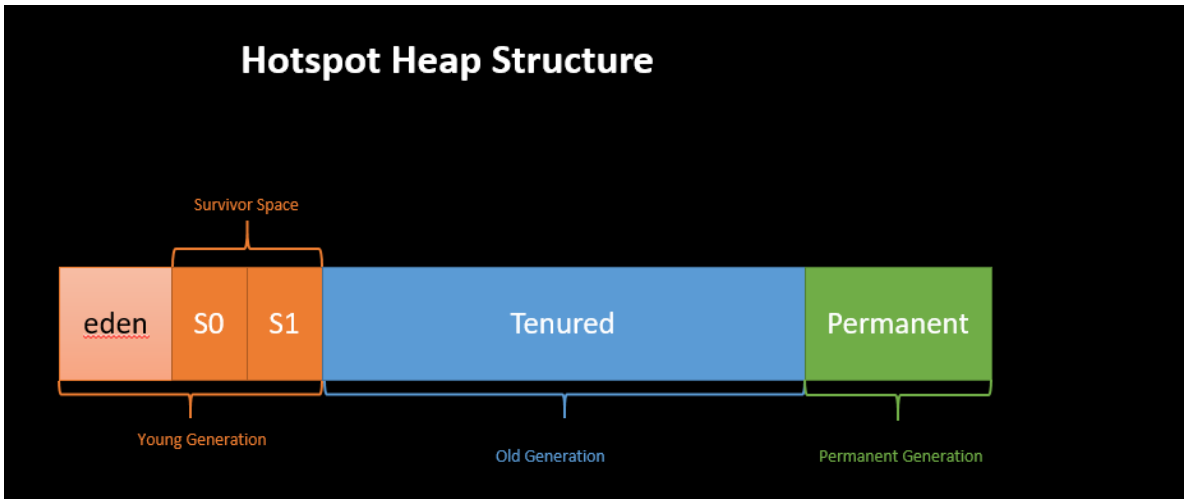
> 有碎片，在创建大对象的时候，可能出现内存不足。

2. `Marking-Compat`(标记-整理法)

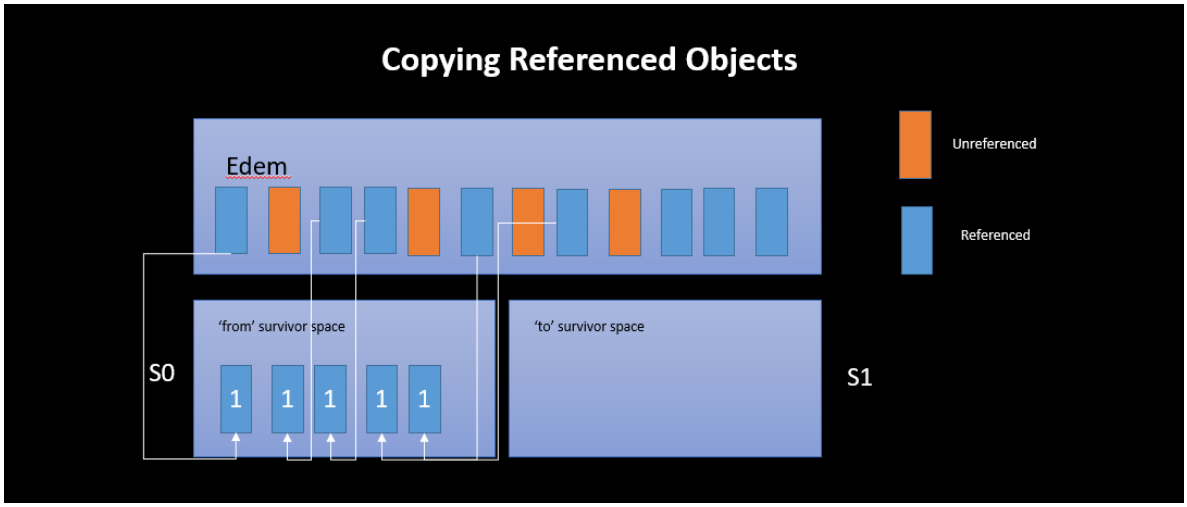
> 没有碎片



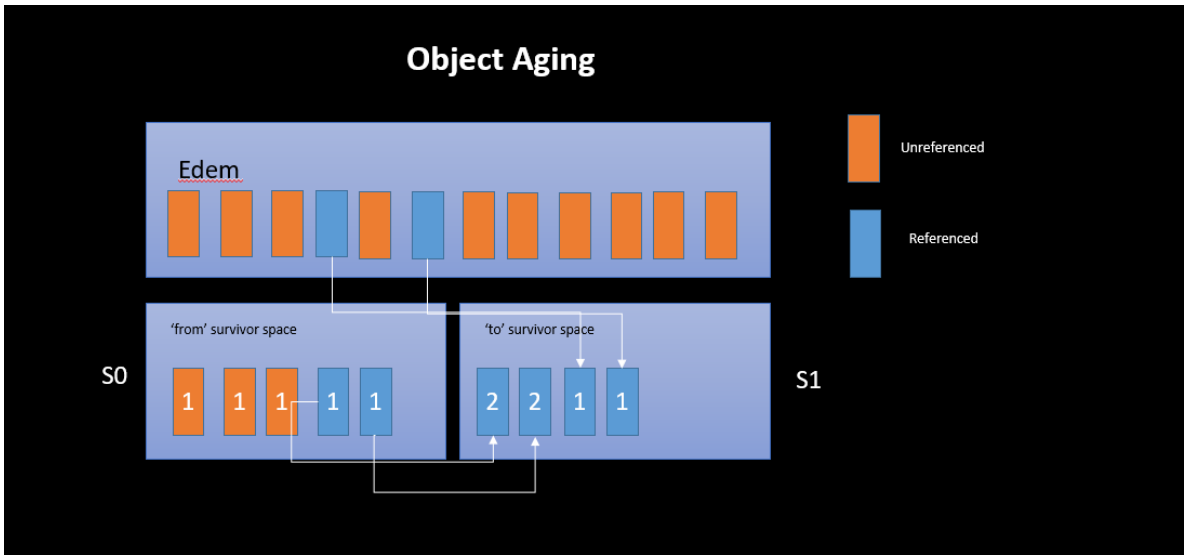
分代



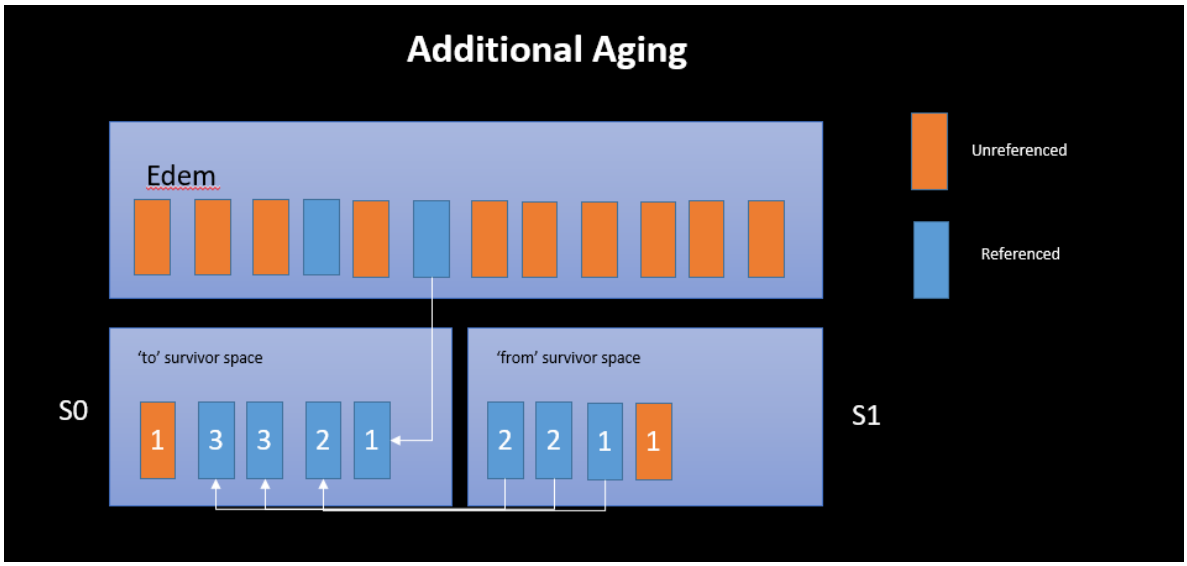
当`edem`空间不足的时候，会把`edem`中的对象移动到`s0`中。并且把这个对象基数为1.



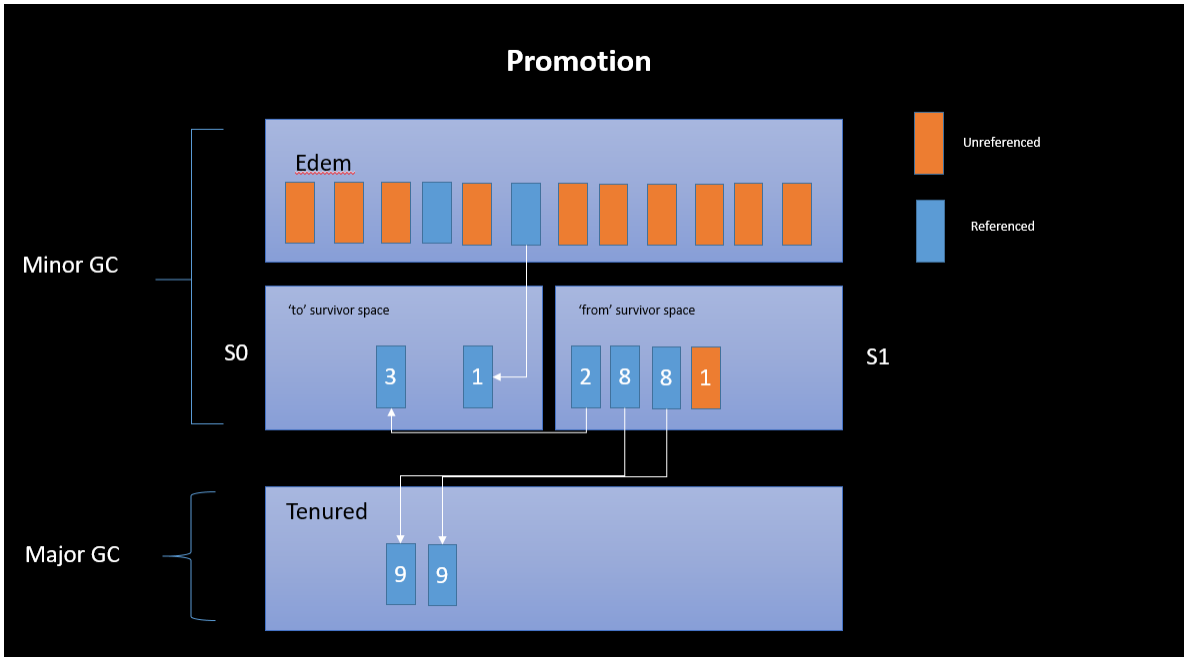
后期继续分配，发现空间依然不足，需要再次移动对象，把`S0`中的对象移动到`S1`中，计数器再次`+1`.



如果又从`S1`返回到`S0`，计数器任然会自动`+1`.



当对象的计数器累计到一个门限值的时候，并且继续分配空间，空间不足时，就会把计数器中的值高的移动到`Tenured Generation`中去。



时期	GC	频率
Yound Generation	Minor GC	GC 频繁
Tenured Generation	Major GC	GC 频率低

常见的回收器

- Seriral Garbage Collector: 单线程GC。
- Parallel Garbage Collector: 多线程GC
- CMS Garbage Collector: 多线程GC
- G1 Garbage Collector: jdk 7 引入GC，多线程，高并发，低暂停。逐步取代CMS GC。

在更高的JRE版本中，G1是默认的GC。