GC（Java Garbage Collection）

GC垃圾收集机制

GC日志信息详解

如何判断对象是否存活

- 引用计数法
- 根搜索方法

GC四大算法

1. 复制算法（Copying）：适用于新生代
2. 标记清除（Mark-Sweep）：适用于老年代
3. 标记压缩（Mark-Compact）：适用于老年代
4. 分代收集算法

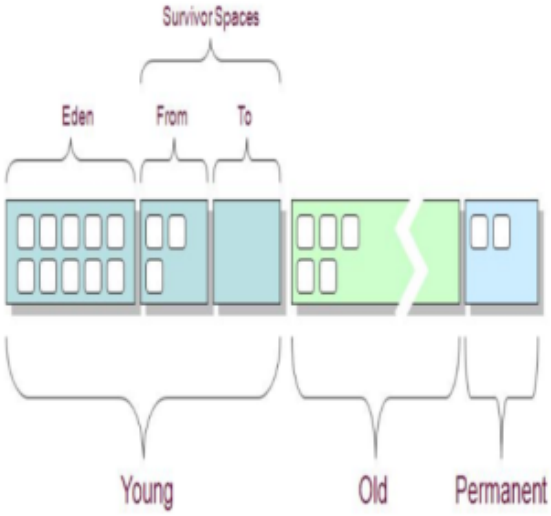
总结

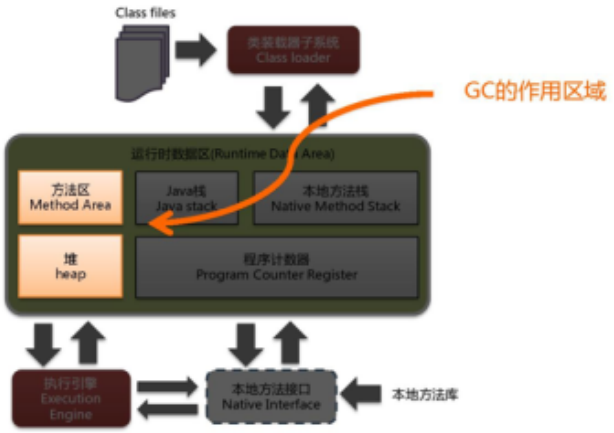
GC常见面试题

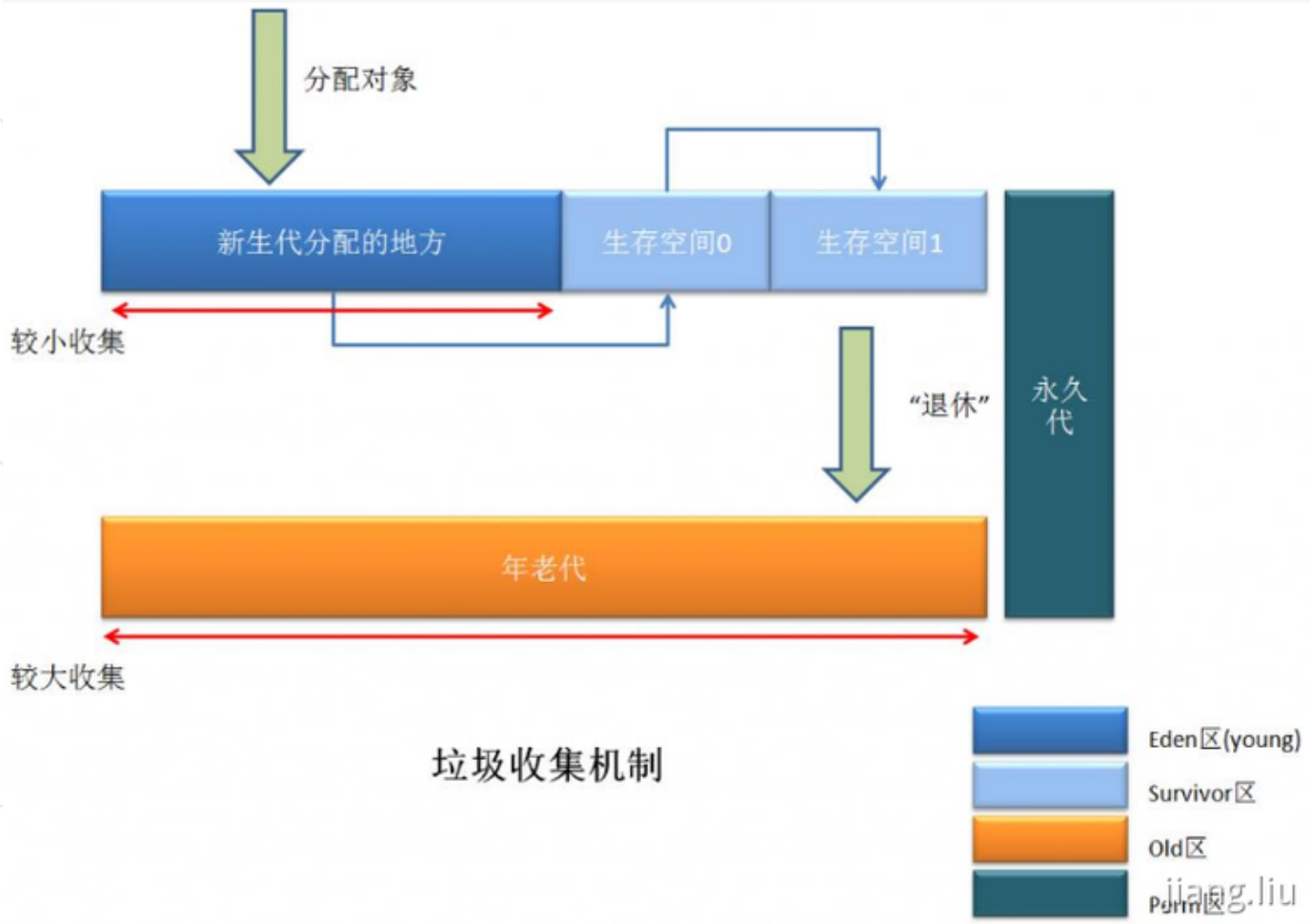
GC垃圾收集机制

对于GC垃圾收集机制，我们需要记住以下几点：

1. 次数上频繁收集Young区。
2. 次数上较少收集Old区。
3. 基本不动元空间。







JVM在进行GC时，并非每次都对上面三个内存区域一起回收的，大部分时候回收的都是指新生代。

因此GC按照回收的区域又分了两种类型，一种是普通GC（minor GC），一种是全局GC（major GC or Full GC）。

Minor GC和Full GC的区别：

（1）普通GC（minor GC）：只针对新生代区域的GC，指发生在新生代的垃圾收集动作，因为大多数Java对象存活率都不高，所以Minor GC非常频繁，一般回收速度也比较快。

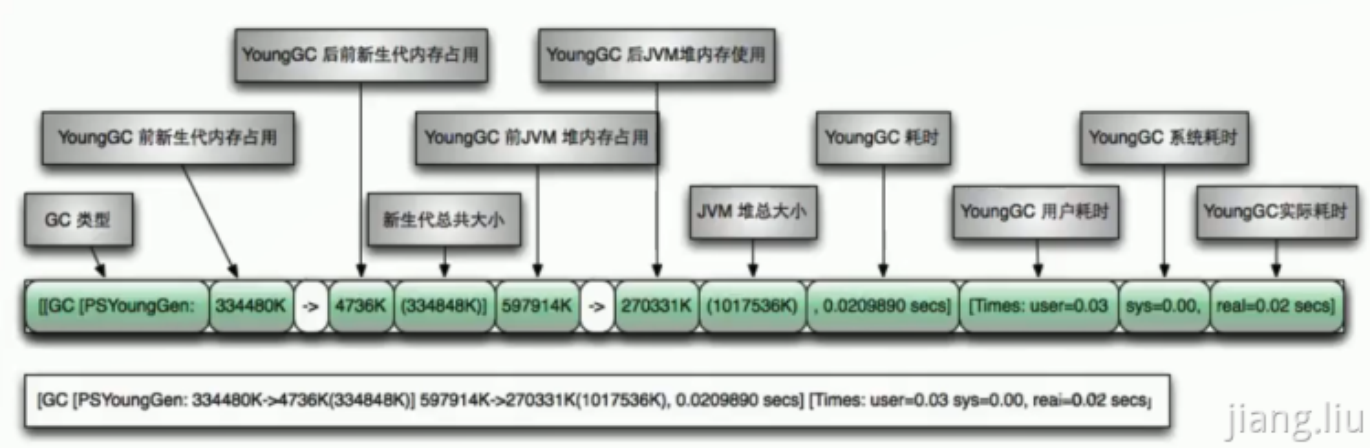
（2）全局GC（major GC or Full GC）：指发生在老年代的垃圾收集动作，出现了Major GC，经常会伴随至少一次的Minor GC（但并不是绝对的）。Major GC的速度一般要比Minor GC慢上10倍以上。

触发Full GC的条件

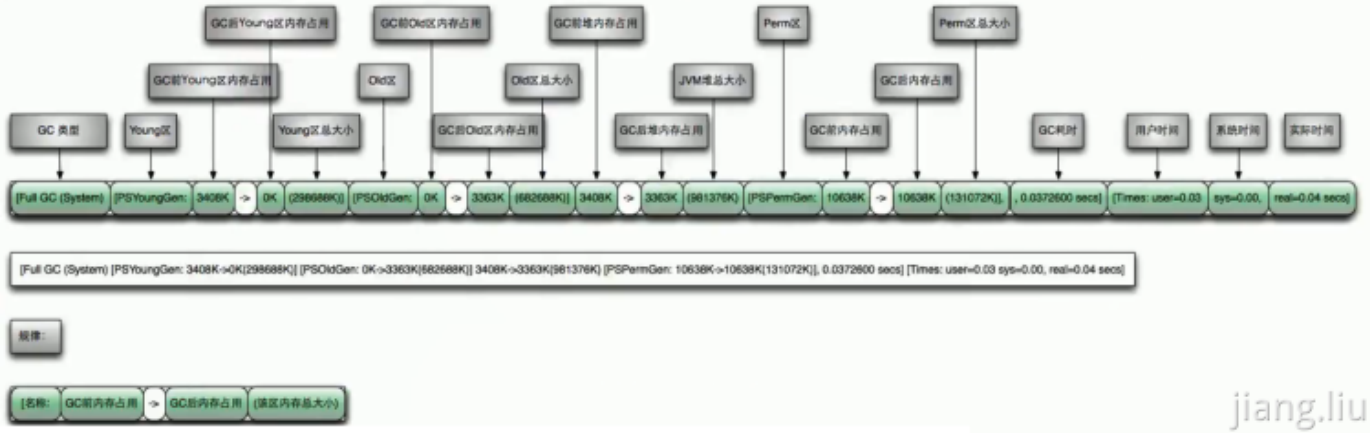
- 老年代空间不足
- 永久代空间不足
- CMS GC时出现promotion failed，concurrent mode failure
- Minor GC晋升到老年代的平均大小大于老年代的剩余空间
- 调用System.gc()

GC日志信息详解

（1）YGC相关参数：



（2）FGC相关参数



如何判断对象是否存活

引用计数法

给每个对象设置一个计数器，当有地方引用这个对象的时候，计数器+1，当引用失效的时候，计数器–1，当计数器为0的时候，JVM就认为该对象不再被使用，是“垃圾”了。

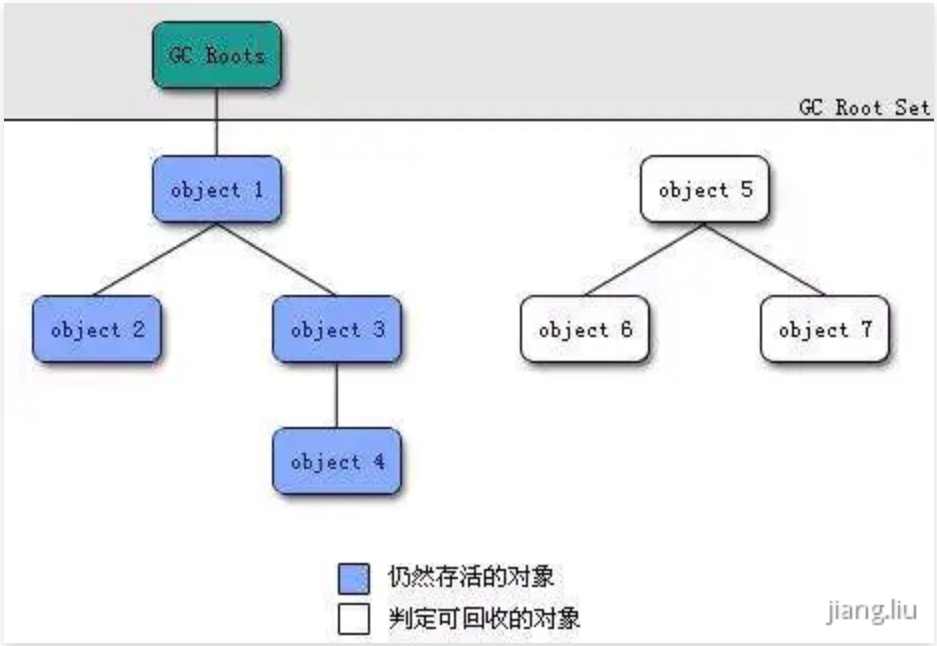
引用计数实现简单，效率高；但是无法解决循环引用问题（A对象引用B对象，B对象又引用A对象，但是A，B对象已不被任何其他对象引用），同时每次计数器的增加和减少都带来了很多额外的开销，所以在JDK1.1之后，这个算法已经不再使用了。

根搜索方法

又叫“可达性分析算法”，通过一些GCRoots对象作为起点，从这些节点开始往下搜索，搜索通过的路径成为引用链（ReferenceChain），当一个对象没有被GCRoots的引用链连接的时候，说明这个对象是不可用的。

GCRoots对象包括：

- 虚拟机栈中引用的对象（栈帧中的本地变量表）。
- 方法区域中的类静态属性引用的对象（譬如Java类的引用类型静态变量）。
- 方法区域中常量引用的对象（譬如字符串常量池里的引用）。
- 方法栈中JNI（`Native`方法）的引用的对象。
- 本地方法栈中JNI(Native方法)引用的对象（非Java语言构建的对象）。
- 活跃线程的引用对象（Java中万物皆为对象，因此活跃的线程的引用对象也会成为GCRoots）。



GC四大算法

- 复制算法
- 标记清除
- 标记压缩
- 分代收集算法

参考资料：https://blog.csdn.net/q961250375/article/details/107859902

1. 复制算法（Copying）：适用于新生代

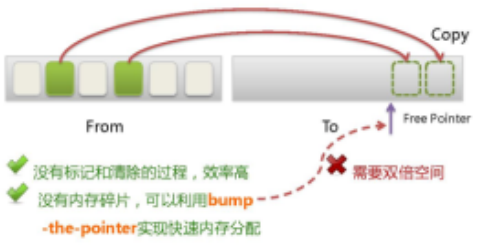
虚拟机把新生代分为了三部分：1个Eden区和2个Survivor区（分别叫from和to），默认比例为8:1:1。

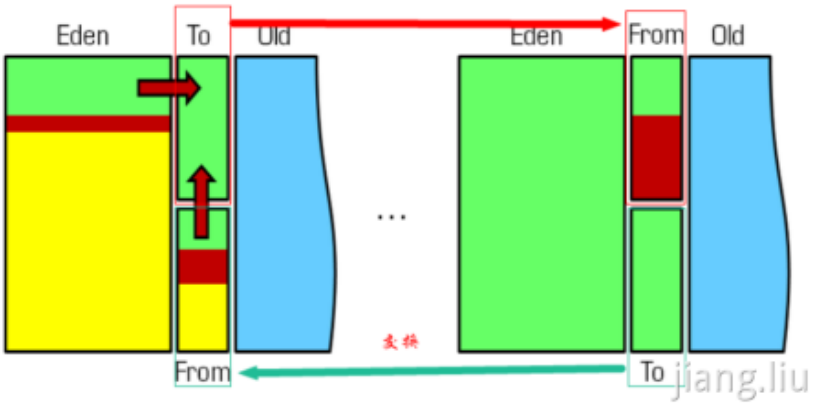
一般情况下，新创建的对象都会被分配到Eden区（一些大对象特殊处理），这些对象经过第一次Minor GC后，如果仍然存活，将会被移到Survivor区。对象在Survivor区中每熬过一次Minor GC，年龄 +1，当它的年龄增加到一定程度时（默认是 15 ，通过-XX:MaxTenuringThreshold来设定参数），就会被移动到老年代中。

因为新生代中的对象基本都是朝生夕死（被GC回收率90%以上），所以在新生代的垃圾回收算法使用的是**复制算法**。

复制算法的基本思想就是将内存分为两块，每次只用其中一块（from），当这一块内存用完，就将还活着的对象复制到另外一块上面。复制算法不会产出内存碎片

我们来举个栗子，在GC开始的时候，对象只会存在于Eden区和名为from的Survivor区，to区是空的。紧接着进行GC，Eden区中所有存活的对象都会被复制到to，而在from区中，仍存活的对象会根据他们的年龄值来决定去向。年龄达到一定值（默认15）的对象会被移动到老年代中，没有达到阈值的对象会被复制到to区域。经过这次GC后，Eden区和from区已经被清空。这个时候，from和to会交换他们的角色，也就是新的to就是上次GC前的from，新的from就是上次GC前的to。不管怎样，都会保证名为to的Survivor区域是空的。Minor GC会一直重复这样的过程，直到to区被填满，to区被填满之后，会将所有对象移动到老年代中。





`-XX:MaxTenuringThreshold` 设置对象在新生代中存活的次数。

因为`Eden`区对象一般存活率较低，一般的，使用两块10%的内存作为空闲和活动区间，而另外80%的内存，则是用来给新建对象分配内存的。一旦发生GC，将10%的`from`活动区间与另外80%中存活的`Eden`区对象转移到10%的`to`空闲区间，接下来，将之前90%的内存全部释放，以此类推。

上面动画中，Area空闲代表to，Area激活代表from，绿色代表不被回收的，红色代表被回收的。

优点 ： 不会产生内存碎片，效率高。
缺点 ： 耗费内存空间。

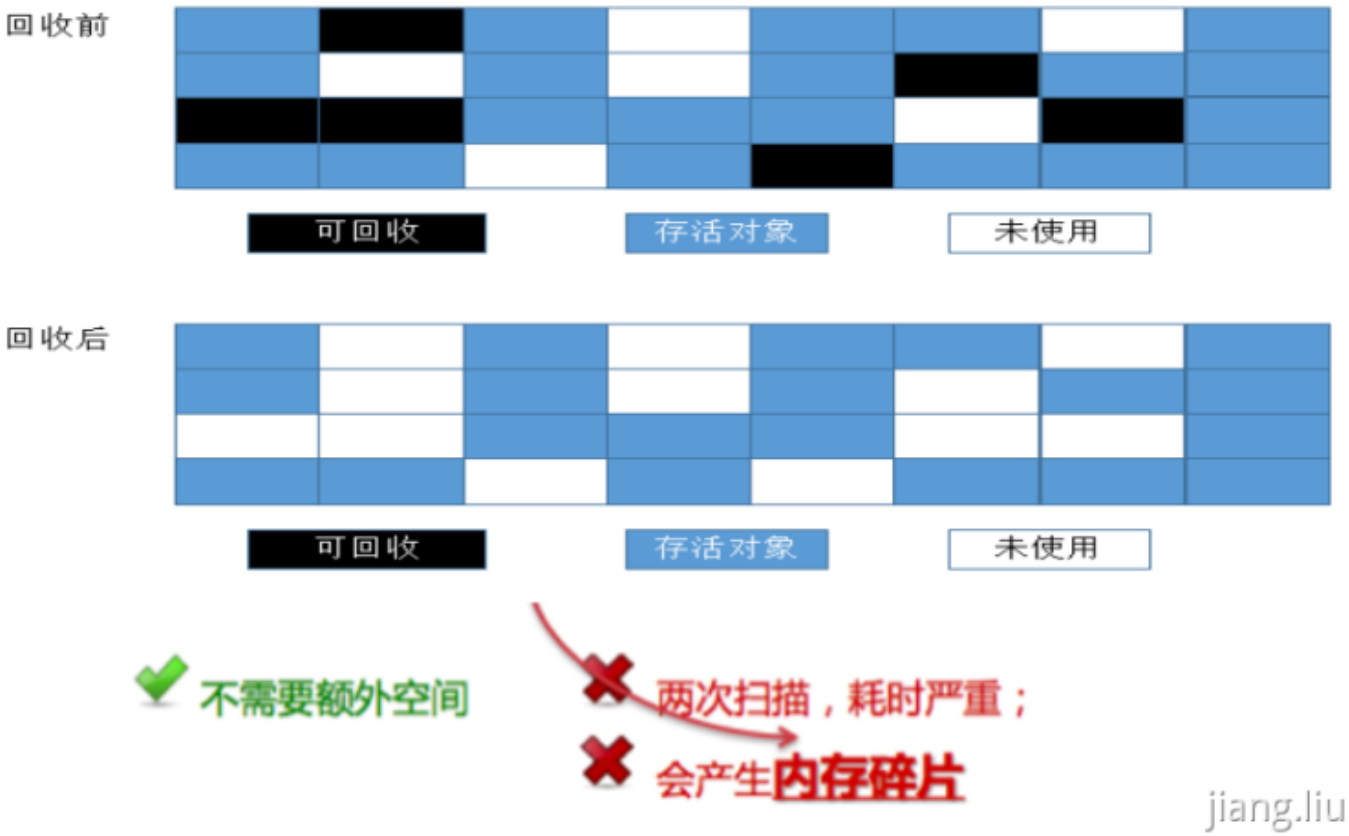
如果对象的存活率很高，我们可以极端一点，假设是100%存活，那么我们需要将所有对象都复制一遍，并将所有引用地址重置一遍。复制这一工作所花费的时间，在对象存活率达到一定程度时，将会变的不可忽视。

所以从以上描述不难看出，复制算法要想使用，最起码对象的存活率要非常低才行，而且最重要的是，我们必须要克服50%内存的浪费。

为什么要有Survivor区？为什么要设置两个survivor区？
https://blog.csdn.net/u012799221/article/details/73180509

2. 标记清除（Mark-Sweep）：适用于老年代

标记清除算法，主要分成标记和清除两个阶段，先标记出要回收的对象，然后统一回收这些对象，如下图:



简单来说，标记清除算法就是当程序运行期间，若可以使用的内存被耗尽的时候，GC线程就会被触发并将程序暂停，随后将要回收的对象标记一遍，最终统一回收这些对象，完成标记清除工作接下来便让应用程序恢复运行。

主要进行两项工作，第一项则是标记，第二项则是清除

- 标记：从引用根节点开始标记遍历所有的`GC Roots`， 先标记出要回收的对象。
- 清除：遍历整个堆，把标记的对象清除。

标记清除算法优缺点：

1、标记和清除两个动作都需要遍历所有的对象，并且在gc时要停止应用运行，对于交互性要求比较高的应用来说就不能满足

> 2、通过标记清除算法整理的内存碎片化比较严重，因为被清除的对象可能存在内存的每个角落，所有清理出来的内存是不连贯的。

为什么标记清除会暂停应用?

因为标记和清除是遍历所有的对象，标记是根据root 引用标记，程序在运行的时候会产生新的对象，会导致标记不准确清除对象的时候也就不准确，会出现存活的对象也被当做垃圾回收，所以必须暂停应用标记和清除；

3. 标记压缩（Mark-Compact）：适用于老年代

标记压缩算法，又叫标记整理算法，它解决了标记清除算法碎片化比较严重的问题。

标记压缩算法分为**标记**、整理。

标记这一步和标记清除算法一致，但是清除不同，在清除垃圾对象的时候并不是简单的清除垃圾对象而是讲存活的对象压缩到内存的一端，然后清理边界以外的垃圾对象，从而解决碎片化严重的问题。

简单来说，就是**先标记，后整理（标记->整理/压缩-> 清除）**，如下图所示：

1. **标记** (Mark) :

与 **标记-清除** 一样。

2. **压缩** (Compact) :

再次扫描，并往一端 **滑动** 存活对象。

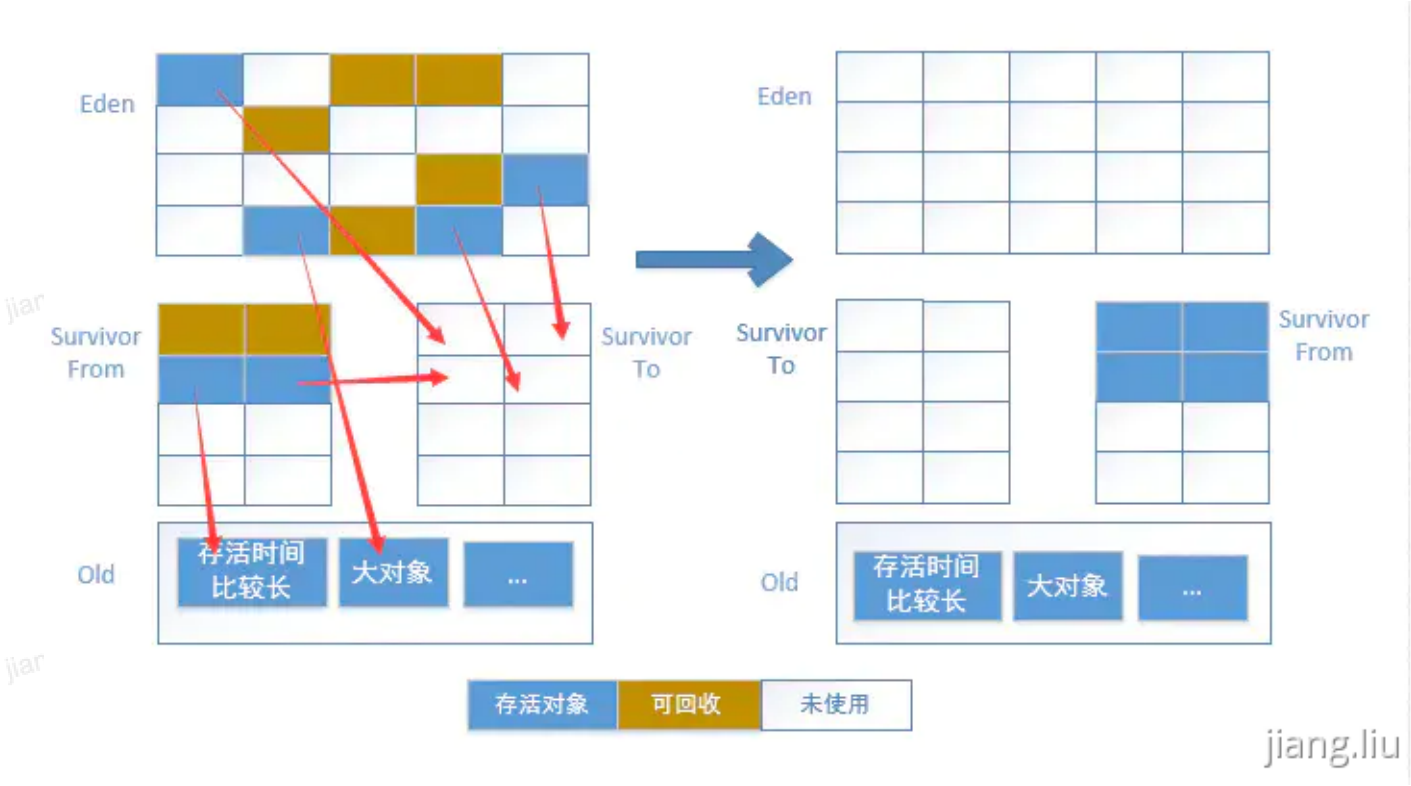
优点 ： 没有内存碎片。

缺点 ： 需要移动对象的成本，效率也不高（不仅要标记所有存活对象，还要整理所有存活对象的引用地址），耗时严重，效率低于复制算法。

4. 分代收集算法

当前商业虚拟机都是采用分代收集算法，它根据对象存活周期的不同将内存划分为几块，一般是把Java堆分为新生代和老年代，然后根据各个年代的特点采用最适当的垃圾收集算法。

在新生代中，每次垃圾收集都发现有大批对象死去，只有少量存活，就选用复制算法，而老年代因为对象存活率高，没有额外空间对它进行分配担保，就必须使用标记清除或者标记压缩算法来进行回收。



总结

1. 年轻代（Young Gen）

年轻代特点是内存空间相对老年代较小，对象存活率低。

复制算法的效率只和当前存活对象大小有关，因而很适用于年轻代的回收。而复制算法的内存利用率不高的问题，可以通过虚拟机中的两个`Survivor`区设计得到缓解。

2. 老年代（Tenure Gen）

老年代的特点是内存空间较大，对象存活率高。

这种情况，存在大量存活率高的对象，复制算法明显变得不合适。一般是由标记清除或者是标记清除与标记整理的混合实现。

（1）标记阶段（Mark） 的开销与存活对象的数量成正比。这点上说来，对于老年代，标记清除或者标记整理有一些不符，但可以通过多核/线程利用，对并发、并行的形式提高标记效率。

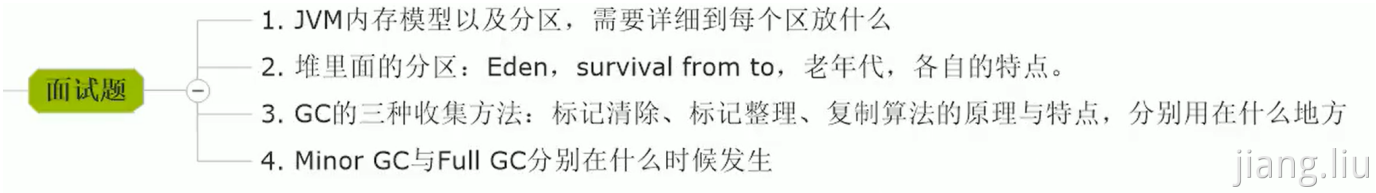
（2）清除阶段（Sweep） 的开销与所管理内存空间大小是正相关。但Sweep“就地处决”的特点，回收的过程没有对象的移动。使其相对其他有对象移动步骤的回收算法，仍然是效率最好的。但是需要解决内存碎片问题。

（3）整理阶段（Compact） 的开销与存活对象的数据成开比。如上一条所描述，对于大量对象的移动是很大开销的，做为老年代的第一选择并不合适。

基于上面的考虑，老年代一般是由标记清除或者是标记清除与标记整理的混合实现。以虚拟机中的CMS回收器为例，CMS是基于Mark-Sweep实现的，对于对象的回收效率很高。而对于碎片问题，CMS采用基于

Mark-Compact算法的Serial Old回收器做为补偿措施：当内存回收不佳（碎片导致的Concurrent Mode Failure时），将采用Serial Old执行Full GC以达到对老年代内存的整理。

GC常见面试题



1. GC四种算法哪个好?

没有哪个算法是能一次性解决所有问题的，因为JVM垃圾回收使用的是分代收集算法，没有最好的算法，只有根据每一代他的垃圾回收的特性用对应的算法。例如新生代使用复制算法，老年代使用标记清除和标记整理算法。

所以说，没有最好的垃圾回收机制，只有最合适的。

2. 请说出各个垃圾回收算法的优缺点?

（1）内存效率： 复制算法 > 标记清除算法 > 标记整理算法（此处的效率只是简单的对比时间复杂度，实际情况不一定如此）。

（2）内存整齐度： 复制算法 = 标记整理算法 > 标记清除算法。

（3）内存利用率： 标记整理算法 = 标记清除算法 > 复制算法。

可以看出，效率上来说，复制算法是当之无愧的老大，但是却浪费了太多内存，而为了尽量兼顾上面所提到的三个指标，标记整理算法相对来说更平滑一些，但效率上依然不尽如人意，它比复制算法多了一个标记的阶段，又比标记清除多了一个整理内存的过程。

难道就没有一种最优算法吗？Java 9 之后出现了G1垃圾回收器（使用分代收集），能够解决以上问题，有兴趣参考这篇文章。