> java的GC机制主要针对堆区中的对象。本文从两个方面描述JavaGC机制
> 1. 如何判定对象应该被回收
> 2. GC算法

一、如何判断哪些对象应该被回收

1.判定对象是否被回收的两种方法

（1）引用计数法

有引用加1，放弃引用减1，当为0时表示可以被回收。

实现简单效率高，但是无法解决对象循环引用的问题

（2）可达性分析

将一系列的称作GCRoot的对象作为跟节点，某个对象到这些GCRoot之间有链路可达，则该对象不应该是被回收的。

> 哪些可作为GCRoot?
> 1. 虚拟机栈栈帧的局部变量表中引用的对象
> 2. 方法区中类的静态属性引用的对象
> 3.方法区中常亮引用的对象
> 4. 本地方法栈中Native方法引用的对象

2. 四种引用方式和区别

（1）强引用

只要强引用存在就不会回收掉被引用的对象，虚拟机宁可抛出OOM

（2）软引用

虚拟机在将要发生OOM之前进行一次针对该类对象的回收，如果此次回收内存够用则好，不够用则会抛出OOM。

（3）弱引用

虚拟机每次发起GC时都会针对该类对象进行回收。无论内存是否紧张。

（4）虚引用

该类引用其实与对象的生命周期并无关系，也无法通过该引用获得该类对象的实例。唯一用处是可以设置一个虚引用关联，当该类对象被回收时可以获得系统通知。

3. 回收过程及对象的自我救赎

一个对象真正死亡，至少要经历两次标记过程。

第一次标记过程中如果发现对象没有雨GCRoot相连，则被标记一次，并查看该对象是否覆盖了finalize()方法，如果覆盖了finalize方法则将该对象加入一个F-Queue的队列中。稍后由虚拟机的Finalizer线程来执行它的方法。但是该线程只是保证该方法有执行机会，并不一定等到它执行完，因为防止里面有死循环导致虚拟机崩溃。

对象可以在finalize方法中将自己与GCRoot相连以拯救自己。

对象的finalize方法只会执行一次。

4. 回收方法区

方法区的回收主要是针对废弃的常量和无用的类。
废弃的常量：该常量没有被任何变量引用
无用的类：
（1）该类的所有实例都已经被回收
（2）该类的classLoder已经被回收
（3）该类对应的java.lang.class对象没有在任何地方被引用，无法通过反射访问该类的任何方法。

二、垃圾搜集算法

1. 标记-清除算法

（1）标记：扫描之后对被认定为需要回收的对象进行回收
（2）清除：对被标记的对象进行GC

- 缺点：
- 标记和清除两个过程效率都不高
- 容易造成内存碎片，不利于后续大对象的内存分配

2.复制算法

（1）将内存分为两块，每次只使用其中的一块。
（2）当其中一块要用完时，标记对象将要存活的对象复制到另外一块上去，然后针对当前块进行整块回收。

- 优点：实现简单高效，无内存碎片。

- 缺点：可用内存变成原来的一半。

3. 标记整理算法

（1）先对将要被回收的对象进行标记
（2）将要存活的对象向内存的一边移动整理
（3）将可存活对象端边界以外的对象进行回收

4. 分代收集算法

将堆划分为新生代和老年代
（1）老年代：采用标记清除或标记整理算法
（2）新生代：采用复制算法

> 新生代分为Eden区和两个survival区，每次使用Eden区和一个survival区，当要回收时将要存活的对象复制survival To区。
> 实验表明，新生代的对象朝生熄灭，一次回收率可达约98%。所以需要复制的对象很少。（注：老年代对新生代的担保机制）

三、内存分配规则

1. 对象优先分配在Eden区

> - 新生代GC：(Minor GC)频繁且速度快
> - 老年代GC：(Major GC)速度比Minor GC慢10倍以上

2. 大对象直接进入老年代：如数组

3. 长期存活的对象进入老年代

新生代的对象每躲过一次GC则增加一岁，当超过默认15岁的时候就认为是长期存活的对象，将进入老年代。

4. 动态对象年龄判定

当新生代中对象没有达到15岁年龄，但是survival区存活的对象中某个相同年龄的所有对象占用的空间超过survival区内存的一半时，将这个年龄及大于这个年龄的对象

移至老年代。

5. 空间分配担保

Minor GC之前虚拟机会检查老年代的剩余空间是否足够容纳新生代存活对象的总空间，如果大于，则安全，否则在允许担保失败的情况下，检查老年代最大可用连续空间是否大于历次新生代晋升到老年代的空间大小，如果大于则进行一次Minor GC（有风险，担保失败的话会进行一次Full GC），如果小于则进行一次Full GC。