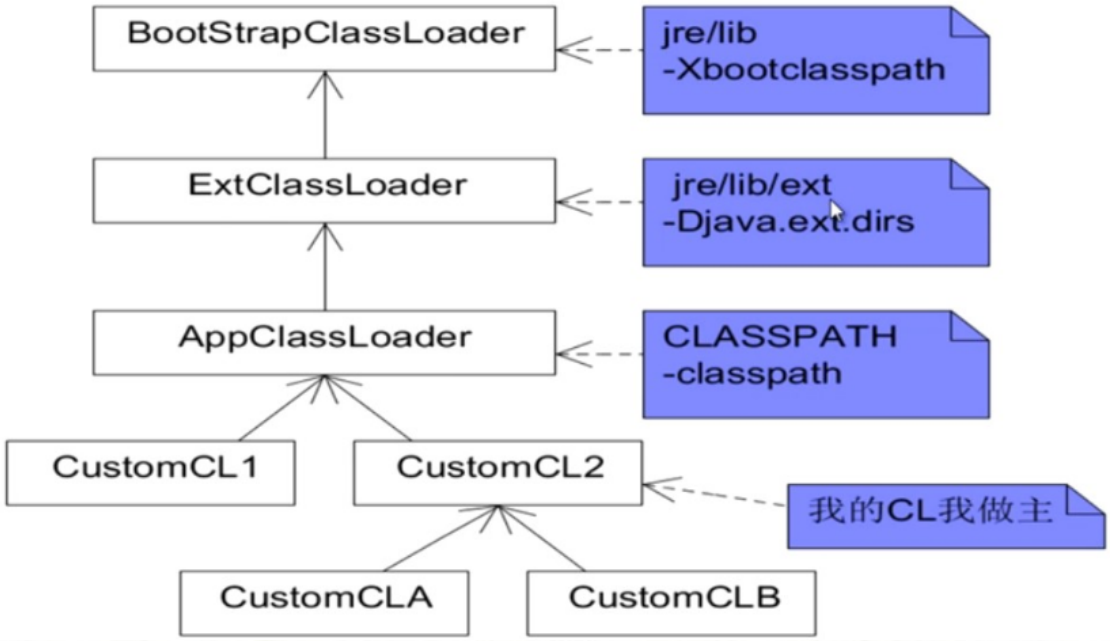
http://www.cnblogs.com/cloudml/p/4713642.html

一、ClassLoader类加载机制

在java中类加载是遵循委派双亲加载的：通过调用loadClass方法逐级往上传递委派加载请求，当找不到父ClassLoader时调用其findClass方法尝试进行查找和加载，如果当前ClassLoad找不所需的Class,则由其孩子尝试进行查找和加载，如果当前ClassLoader找了所需的Class则将该Class按请求路径逐级返回孩子。其关系图如下所示：



ClassLoader.loadClass(...) 是ClassLoader的入口点。当一个类没有指明用什么加载器加载的时候，JVM默认采用AppClassLoader加载器加载没有加载过的class，调用的方法的入口就是loadClass(...)。如果一个class被自定义的ClassLoader加载，那么JVM也会调用这个自定义的ClassLoader.loadClass(...)方法来加载class内部引用的一些别的class文件。重载这个方法，能实现自定义加载class的方式，抛弃双亲委托机制如果要实现自己的类加载器，只需继承ClassLoader，重写findClass方法。

二、内存管理

1.JVM内存模型

（1）方法区：存储类的结构信息，类、类加载器元数据----永久代（堆的一部分），运行时常量池（每个class运行时的常量表）存储已被JVM加载的类信息、常量、静态变量、即时编译器编译后的代码等数据 —XX:PermSize、—XX:MaxPermSize

（2）堆：存放Java对象,数组

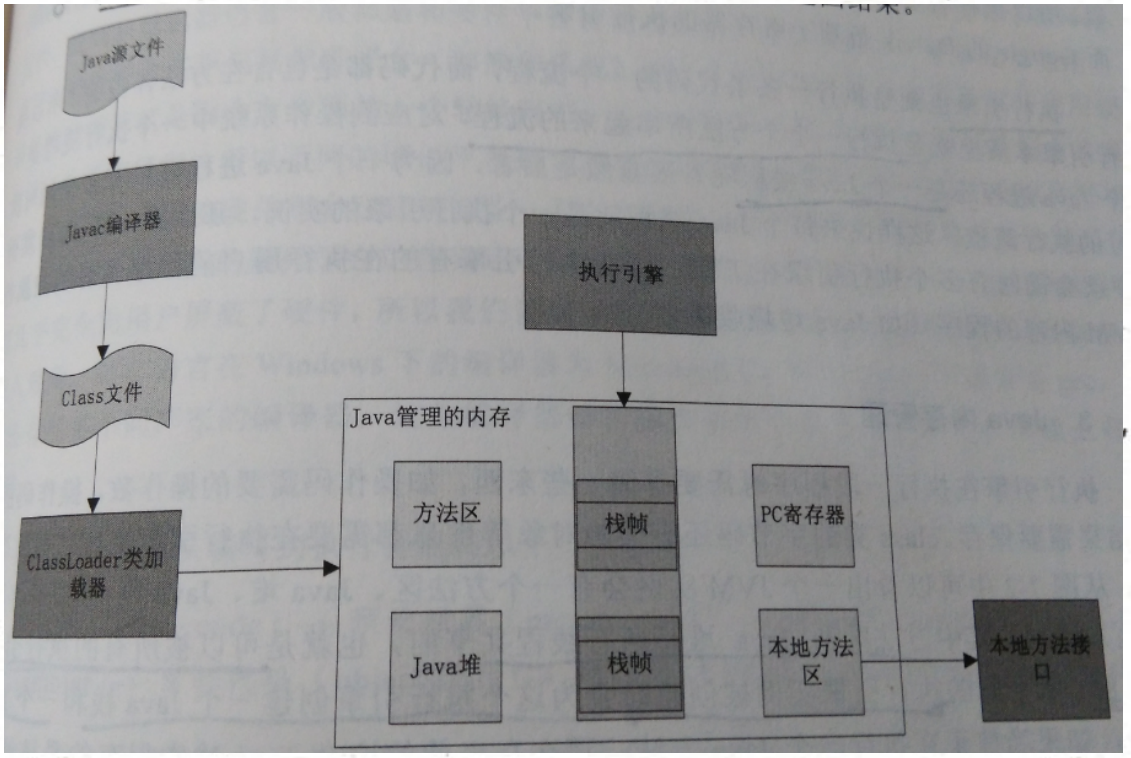
（3）栈：执行引擎：执行一个个方法的串行流程，是线程。每当创建一个线程时为其创建一个java栈和pc计数器，每调用一个方法则在栈中创建一个栈帧（保留方法的元信息：局部变量，返回地址等）

（4）本地方法栈：运行native方法的存储空间

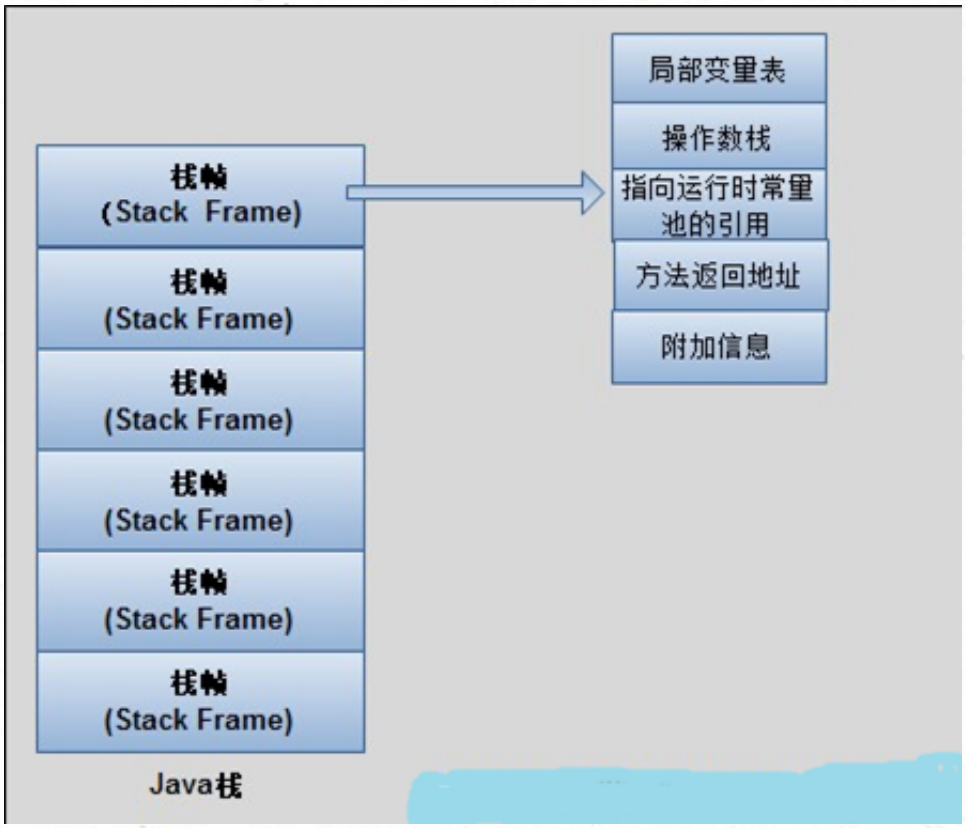
（5）静态分配：编译时已确定（局部变量（含原始数据类型）、对象引用（指向对象在堆中的地址）），栈帧分配，方法结束则消失

（6）动态分配：程序执行时才确定，创建java对象时在堆分配 可共享 方法结束不一定消失， 内存回收已对象不再引用（直接或间接）为前期

Java文件从源文件到在JVM执行的流程如下：



JAVA栈结构图:



2.内存回收依据

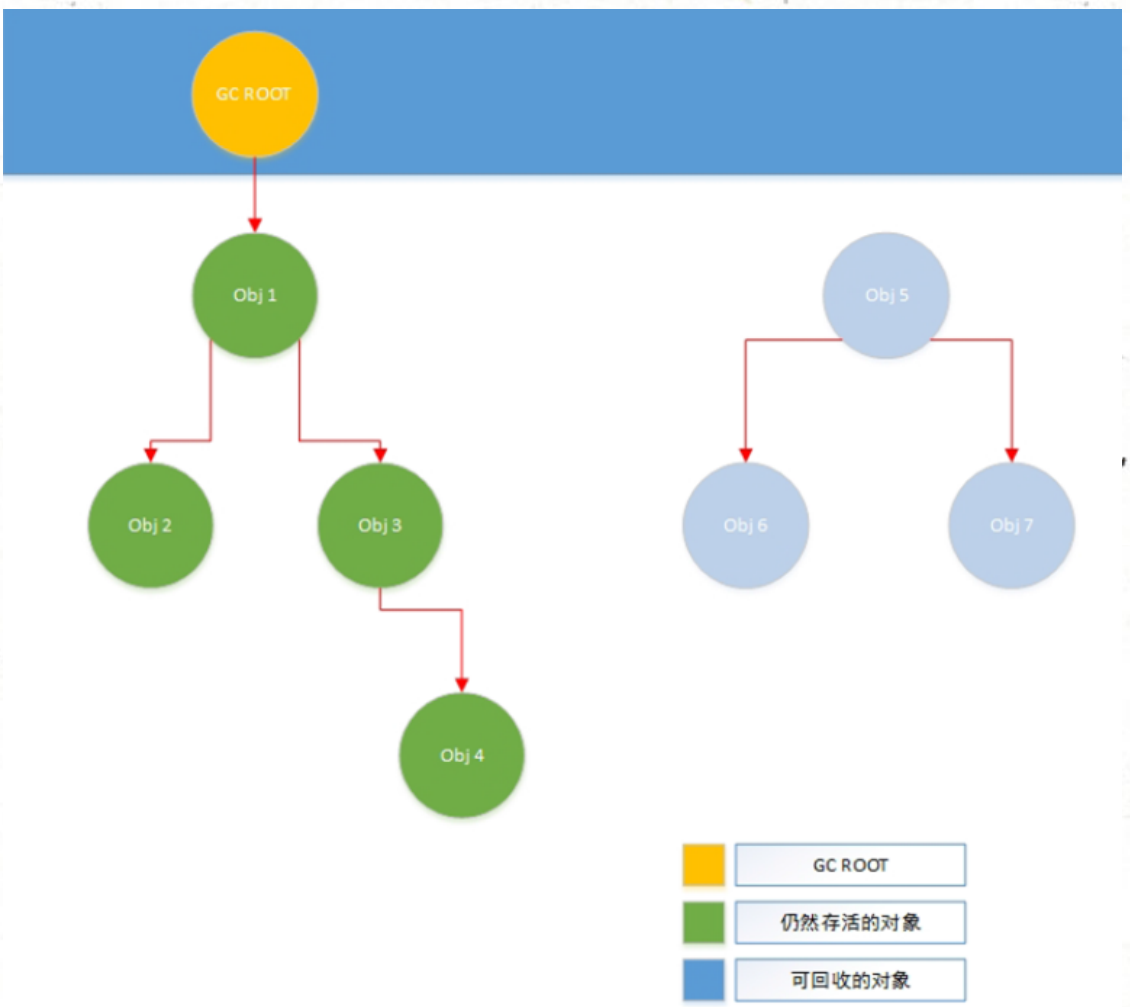
判断对象可回收的依据是对象不再被引用，主要有两种方法:

（1）计数法

给对象添加一个引用计数器，每当该对象被引用，它的计数器值就＋1；当引用失效时，计数器就－1；在任何情况下，当计数器值为0时，就表示该对象不再被使用

（2）可达性分析

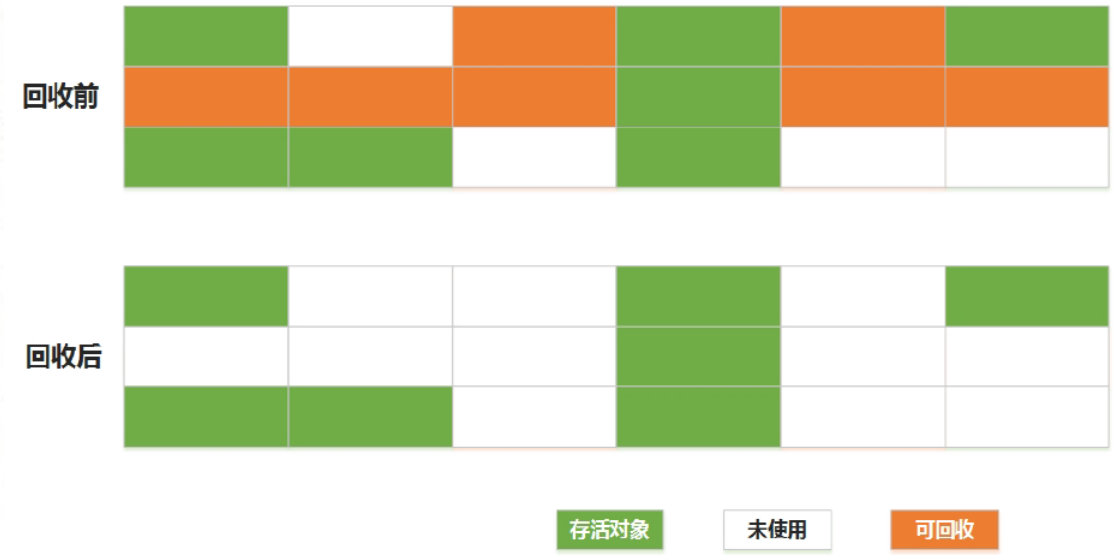
只要改对象不再被其他活动对象引用就可回收，活动对象是指从gc根对象有路径达到该对象。gc根对象即对象的引用



3.回收算法

（1）标记清除算法

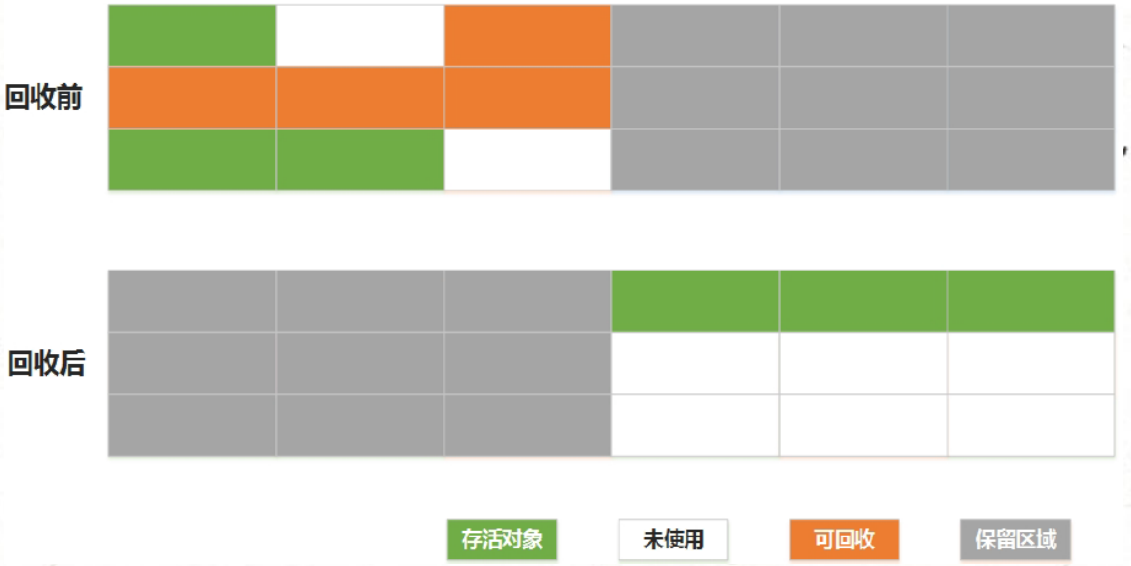
首先标记出所有需要回收的对象，而后在标记完成后统一回收所有被标记的对象。存在.效率问题，标记和清除两个过程的效率都不高； 空间碎片问题，标记清除后会产生大量不连续的内存碎片，空间碎片太多可能会导致以后在程序运行中需要分配较大对象时，无法找到足够的连续内存而不得不提前触发另一个垃圾回收动作。



（2）复制算法

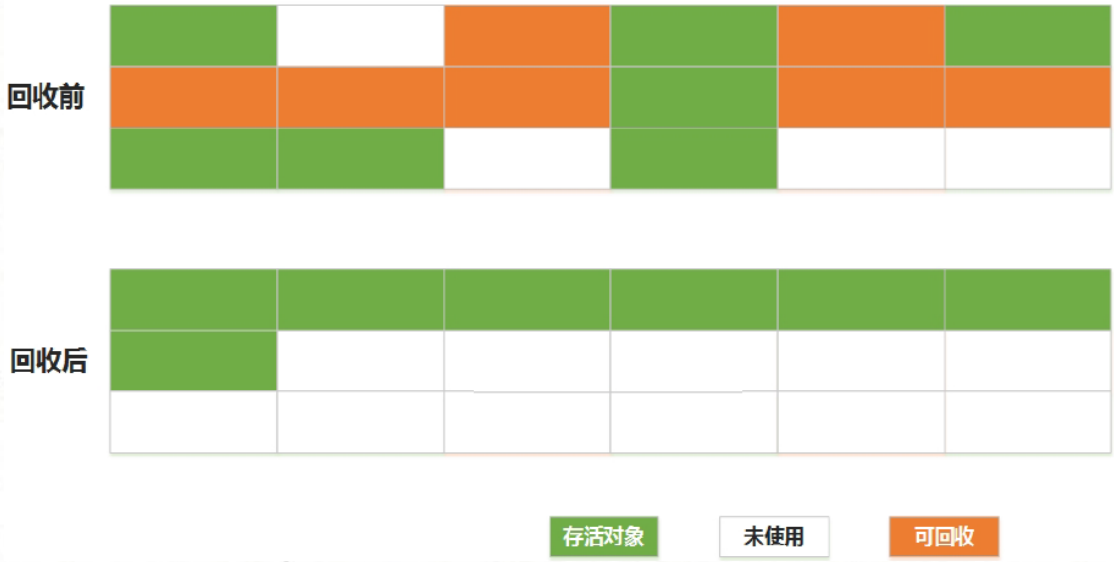
将内存划分为相等的两块，每次只使用其中一块。当这一块内存用完时，就将还存活的对象复制到另一块上面，然后将已经使用过的内存空间一次清理掉。

缺点：将内存缩小为了原来的一半，对内存空间耗费较大。在对象存活率较高时，需要进行多次复制操作，效率会变低。



（3）标记整理算法

不直接对可回收对象进行清理，而是让所有存活对象都向另一端移动，然后直接清理掉端边界以外的内存。



（4）分代收集算法

把对象按照寿命长短进行分组，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收集算法，在新生代采用复制算法，在老年代采用"标记－清除"或者"标记－整理"算法。

JVM将整个堆划分为Young区、Old区和Perm区，分别存放不同年龄的对象。

-Xms:堆起始大小

-Xmn:堆的最大大小，通常-Xms，-Xmn:设为一样大，避免后期的堆收缩

-XX:SurvivorRatio：新生代中Eden区域与Survivor区域的容量比值，默认为8:1

-XX:PretenureSizeThreshold：新生代直接晋升到老年代的对象大小，大于这个参数的对象将直接在老年代分配

-XX:MaxTenuringThreshold：晋升到老年代的年龄。一个对象坚持过一次Minor GC后，年龄就增加1，当超过阀值就进入老年代

Young区分为Eden区和两个相同大小的Survivor区，其中所有新创建的对象都分配在Eden区域中，当Eden区域满后会触发minor GC 将Eden区仍然存活的对象复制到其中一个Survivor区域中，另外一个Survivor区中的存活对象也复制到这个Survivor区域中，并始终保持一个Survivor区时空的。

Old区存放Young区Survivor满后触发minor GC后仍然存活的对象，当Eden区满后会将存活的对象放入Survivor区域，如果Survivor区存不下这些对象，GC收集器就会将这些对象直接存放到Old区中，如果Survivor区中的对象足够老，也直接存放到Old区中。如果Old区满了，将会触发Full GC回收整个堆内存。

Perm区主要存放类的Class对象和常量，如果类不停地动态加载，也会导致Perm区满。Perm区地垃圾回收也是有Full GC触发地。

新创建的对象被分配在新生代，如果对象经过几次回收后仍然存活，那么就把这个对象划分到老年代。老年代的收集频度不象年轻代那么频繁，这样就减少了每次垃圾回收所需要扫描的对象，从而提高了垃圾回收效率。

Serial收集器是一个单线程收集器，它进行垃圾收集时，必须暂停其他所有的工作线程（Stop the world），直到它垃圾收集结束 client模式。
ParNew收集器其实就是Serial收集器的多线程版本，在运行在Server模式下的虚拟机中，ParNew收集器是首选的新生代收集器。
CMS收集器是一款并发收集器（用户线程与垃圾收集线程同时执行），是一种以获取最短回收停顿时间为目标的收集器，它是基于标记-清除算法实现的初始标记、重新标记仍然需要"Stop the World"，但是它们的速度都很快。初始标记只是标记一下GC Roots能直接关联到的对象，重新标记是为了修正并发标记期间因为用户线程继续运作而导致标记产生变动的那一部分对象的标记记录。（初始标记→并发标记→重新标记→并发清除）