java的内存管理全部交由JVM代替实现，开发者不需要关注与内存的分配和回收

1:JVM内存划分

2:对象访问

3:GC－垃圾对象判定策略

4:GC－垃圾回收算法

5:GC－垃圾回收器

6:OOM解决思路

1:JVM内存划分

JVM按照存储内容的不同，将整个JVM内存分为5大类。其中线程私有的有：方法计数器、虚拟机栈、本地方法栈；线程共享的有：方法区、堆。

1:程序计数器

记录线程即将执行的指令。多线程切换时，通过它来保证线程有序的向下执行。

2:虚拟机栈

又名线程方法调用栈。每个栈桢存储方法的入口、引用参数、局部变量表、方法返回地址、操作数栈等。栈桢的入栈和出栈，对应着方法的调用和返回。

3:本地方法栈

与虚拟机栈类似，只不过本地方法栈记录本地方法，而线程方法栈记录java方法。

4:方法区

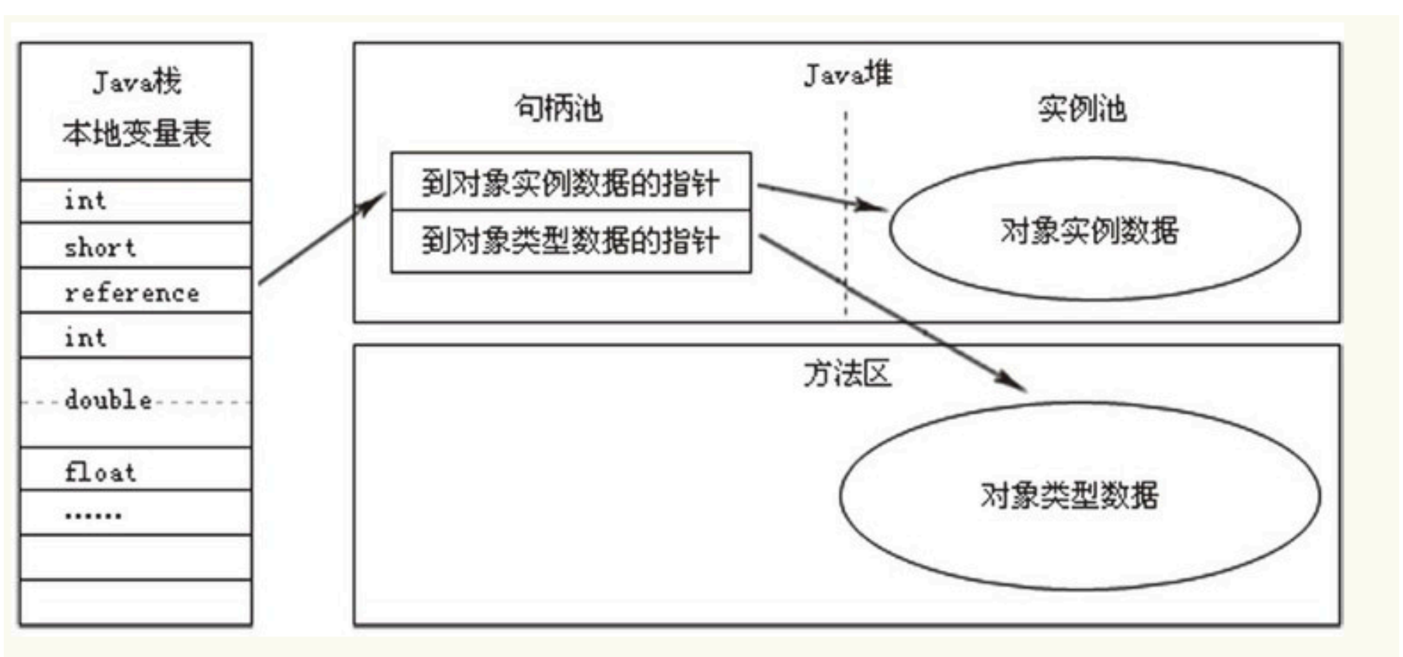
存放Class对象、静态变量、常量、字符串常量等等。

5:堆

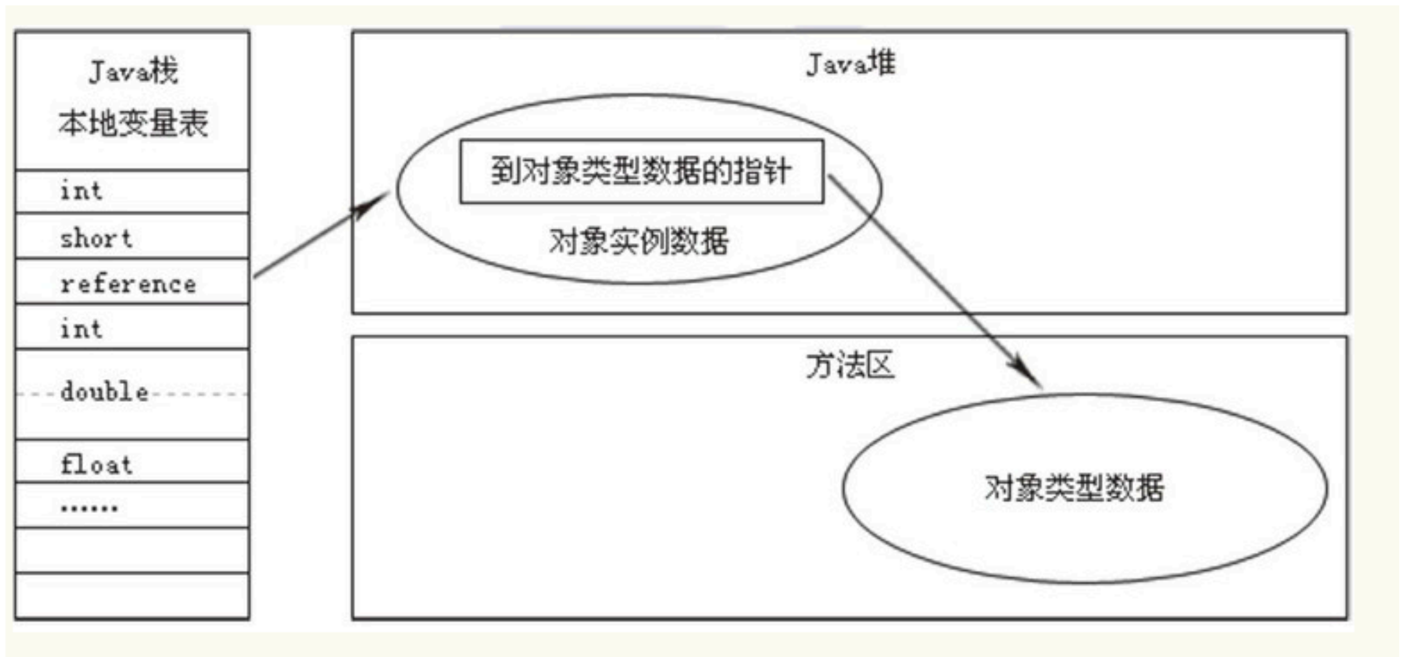
存放对象的区域，根据GC回收的不同策略，又分为老年代和新生代。

2:对象访问

1:使用句柄池



2:直接引用



3:二者比较

使用句柄池，当GC移动对象时，并不需要修改引用，只需修改句柄池中的引用

直接使用，1:访问快，2:节省了句柄池的空间

4:创建对象 A a = new A();

1:在栈中创建一个引用变量a

2:在堆中创建A的实例变量

  2.1:在方法区中找到A的Class对象

  2.2:根据Class对象创建一个实例对象，存放在堆中

3:将新建的对象的地址赋值給变量a

3:GC－垃圾对象判定策略

JVM会自动回收垃圾对象，那么何为垃圾对象呢？计算垃圾对象有以下两种算法

1:引用计数器

记录本对象被引用的次数。当数量为0即可视为垃圾对象。

优点：实现方便，判定效率高

缺点：无法处理循环引用的情况

2:可达性分析

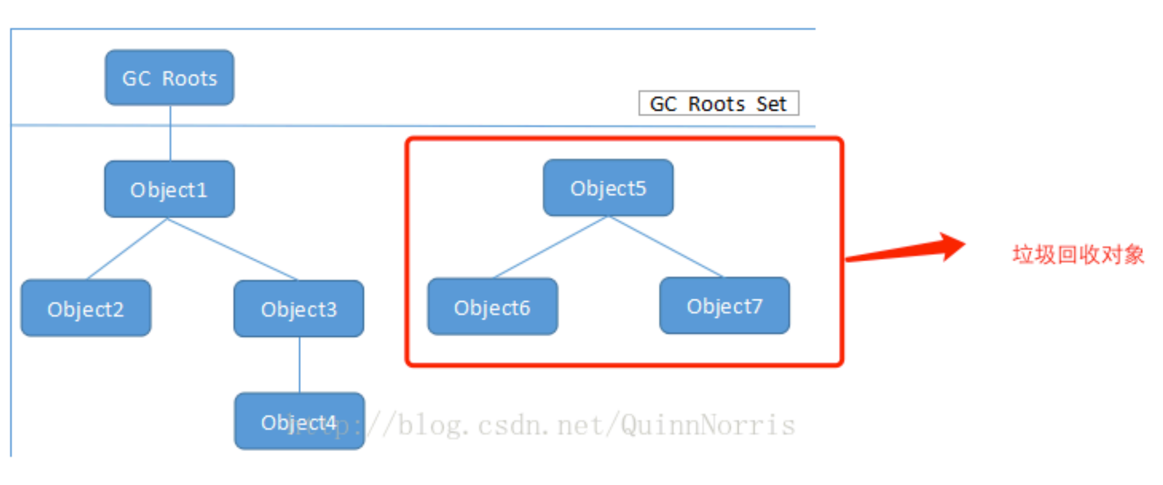
以GC Roots对象为出发点，计算一条对象引用链，不在该链路上的对象即视为垃圾对象。

1:优点：可解决循环引用的情况

2:缺点：实现成本高，效率不及引用计数器

3:GC Roots对象的类型

  虚拟机栈中的变量、方法区中的静态变量、常量



4:GC－垃圾回收算法

4.1:标记－清除法

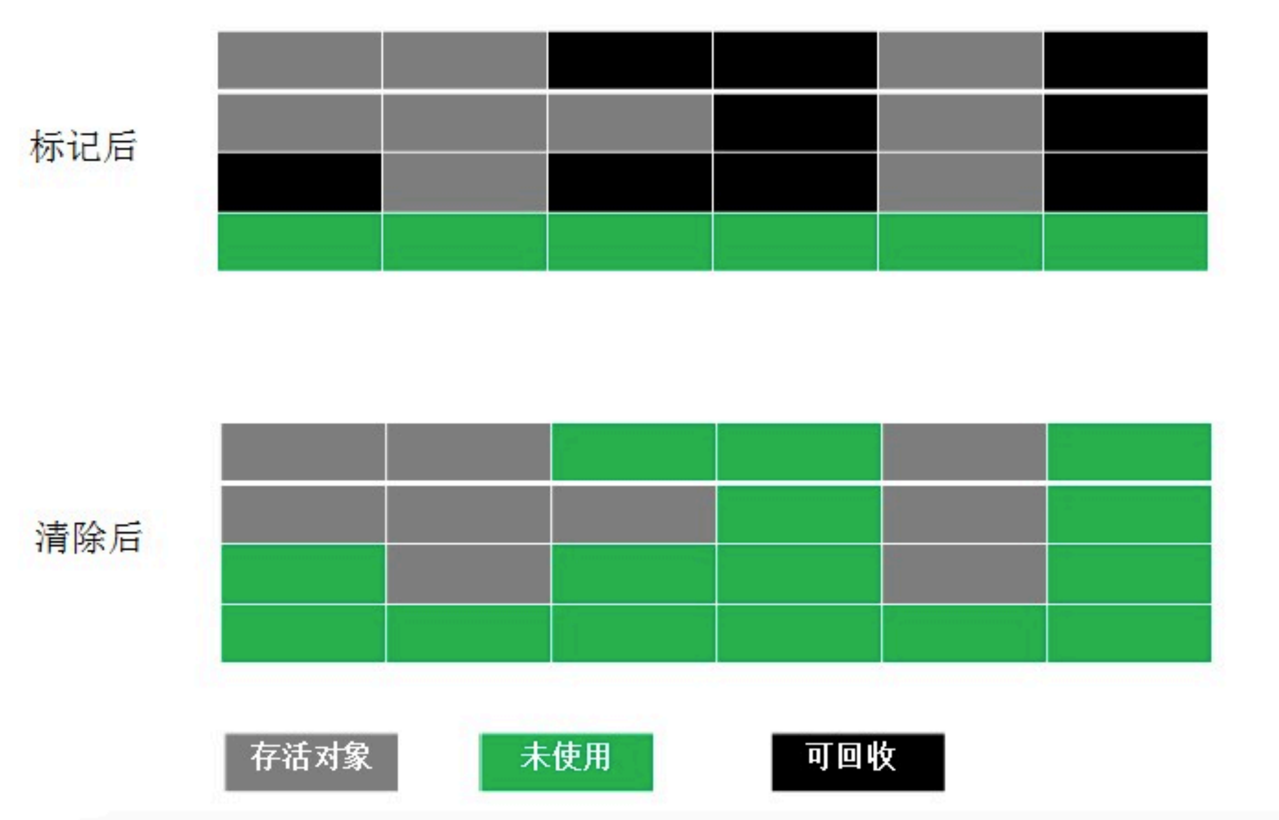
首先找出垃圾对象，然后将对象占用的内存释放

优点

实现方便

缺点

产生大量的内存碎片



4.2:标记－整理法

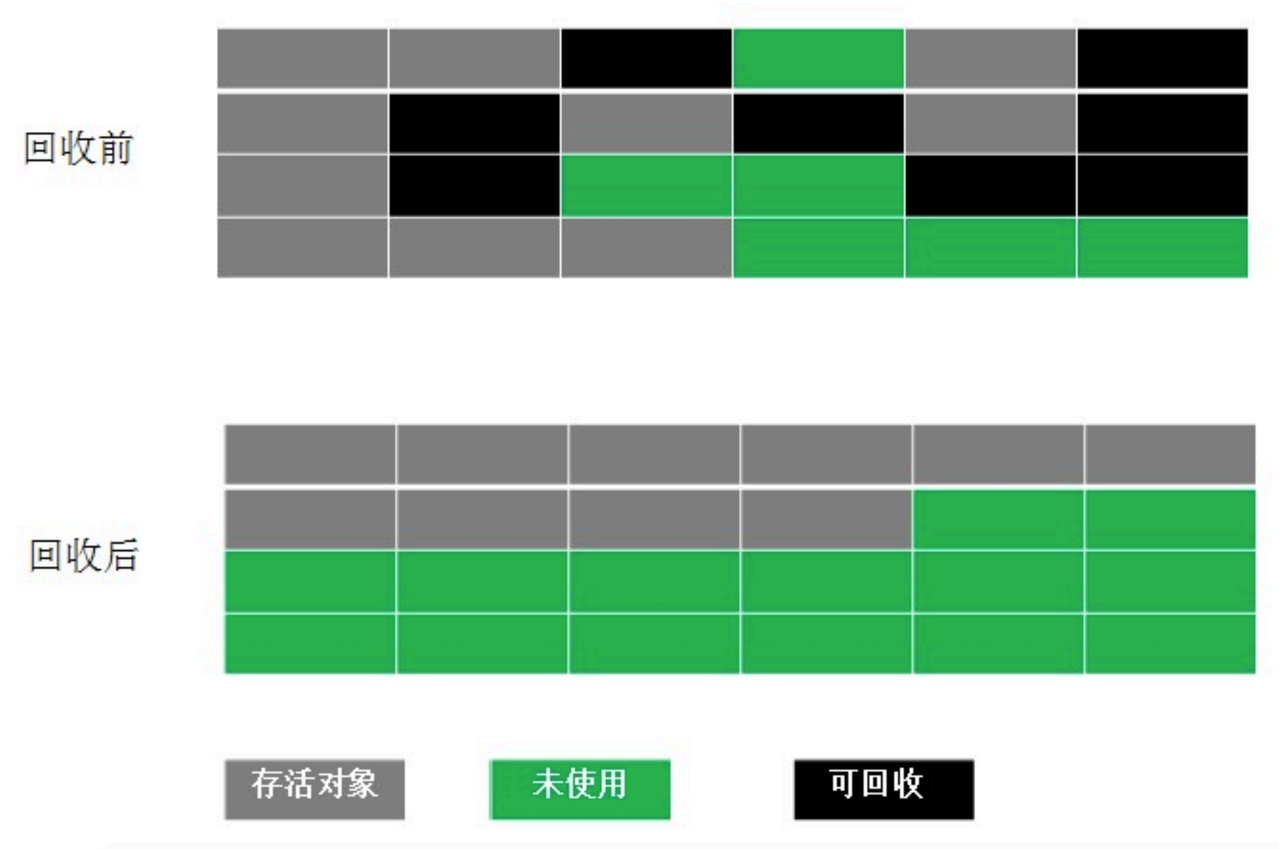
首先找出垃圾对象，然后将垃圾对象占用的内存释放，最后将存放的对象向内存的一边整体移动

优点

不会产生内存碎片，内存空间紧凑

缺点

移动对象成本高



4.3:复制算法

步骤

1:将整个内存分为两块

2:找出存活的对象

3:将存活的对象复制到另外一个内存空间

4:清空本内存空间

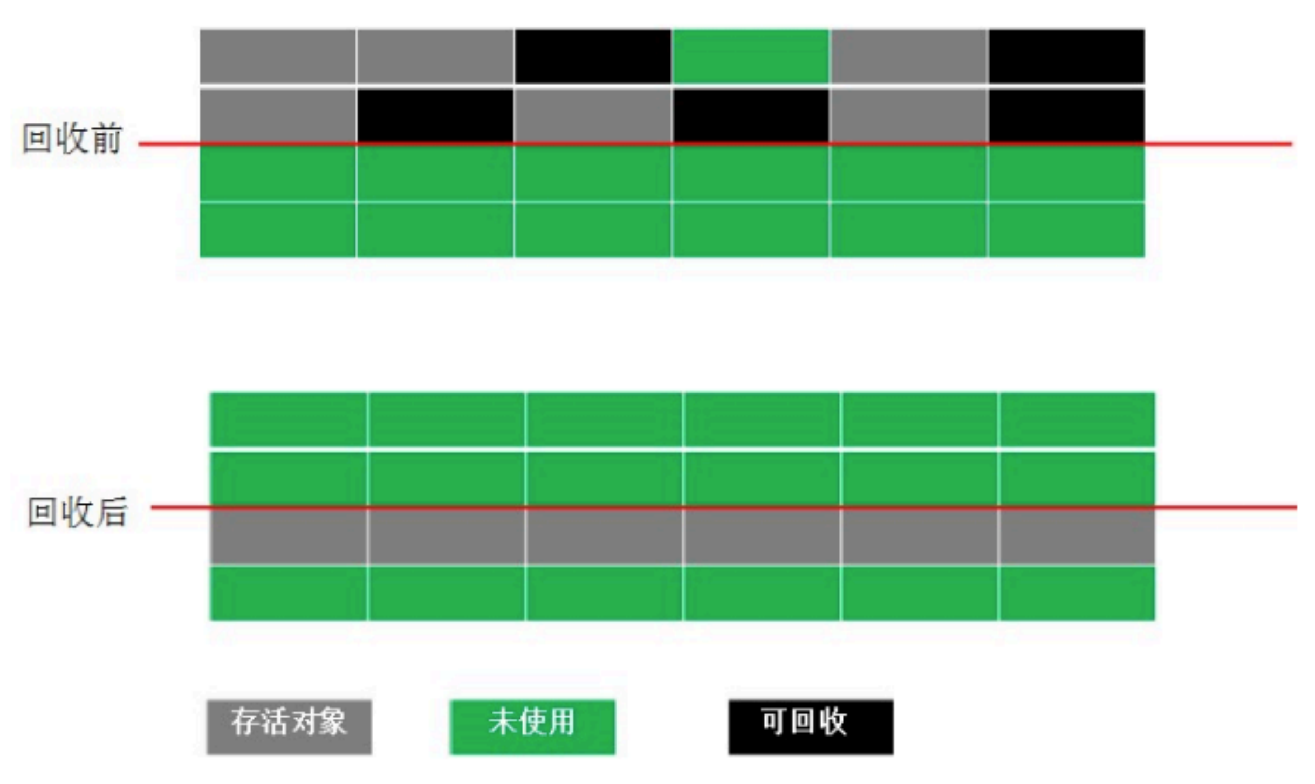
5:反复执行2-4

优点

缓解了对象移动带来的损耗

缺点

内存使用率不高，会浪费一般的内存



4.4:分代算法

将堆按照对象的存活时间分为不同的区域，不同区域使用不同的垃圾回收器

1:将堆分为老年代和新生代

2:将新生代氛围Eden区和survivor

5:GC－垃圾回收器

注释：

STW：Stop The word。即在进行垃圾回收时，所有工作线程暂停工作。

1:Serial

新生代、单线程、STW、复制算法

2:ParNew

新生代、多线程、STW、复制算法

3:Parallel Scavenge

新生代、多线程、STW、复制算法

追求吞吐量，即工作线程工作时间和总的时间的占比。

4:Serial Old

老年代、单线程、STW、标记－整理法

5:Parallel Old

老年代、多线程、STW、标记－整理法

6:CMS:Concurrent Mark Sweep

老年代、多线程、并发清除、标记－清除法

步骤

1:初始标记

简单标记GC Roots对象，该阶段停止工作线程

2:并发标记

对初始标记对象进行可达性分析，搜索存活对象。工作线程不停止

3:重新标记

修改并发阶段状态变化的对象

4:并发清除

清除垃圾对象，工作线程不停止

总结

将比较耗时的并发标记和并发清除，和工作线程共同运行。减少STW停顿时间

优点

GC线程和工作线程并发工作，不会影响系统的交互性

缺点

1:GC线程和工作线程竞争CPU

2:标记－清除算法产生内存碎片

3:会产生浮动垃圾，无法回收并发线程产生的垃圾

7:G1:Garbage-Frist

堆空间、多线程、并发、标记－整理法

思路

将整个对分为若干个region。垃圾回收时按照“垃圾对象空间”排序region,优先清空垃圾对象多的region。

步骤：

初始标记－并发标记－重新标记－并发清除

优点：

有效时间内尽可能多的回收垃圾对象。提高垃圾回收率

详情：

http://www.importnew.com/15311.html

6:OOM解决思路

1:确定OOM发生的区域

利用工具获取JVM内存快照、分析内存快照

1:分析各个类对应的实例变量的个数

2:分析Class数量是否过于庞大

3:线程执行情况，是否有线程长期阻塞

2:方法区

1:加载的类太多，可以考虑分布式拆分系统

2:可以考虑扩容JVM内存

3:注意字字符串常量

3:虚拟机栈

1:考虑栈容量是否太小

2:考虑是否有大量的递归调用

4:堆

1:根据内存快照分析是否有内存泄漏的情况

2:分析容量