# 垃圾回收

## 垃圾回收概述

### 什么是垃圾

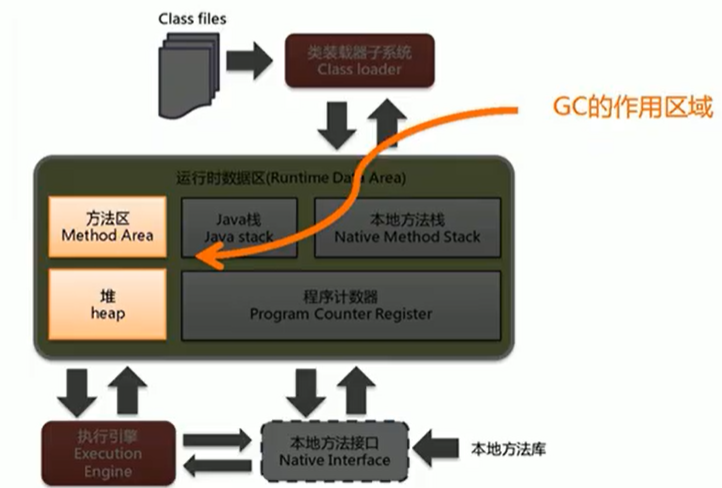
垃圾是指在**运行程序中没有任何指针指向的对象**。

### 为什么要有GC？

1. 内存迟早会被消 耗完
2. 清除内存里的记录碎片，JVM将整理出内存分配给新的对象
3. 没有GC就不能保证应用程序的正常进行

### JAVA垃圾回收机制

自动内存管理，**降低内存泄漏和内存溢出**的风险



## 垃圾回收算法

### 垃圾标记阶段算法-引用计数算法

垃圾标记阶段：区分出内存中哪些是存活对象，哪些是死亡对象，只有被标记为死亡的对象，最后才会被释放所占的内存空间。

引用计数算法（Reference Counting）：对每个对象保存一个**整型的引用计数器属性**，用于记录对象被引用的情况。

只要被引用，计数器+1，引用失效，计数器-1，当为0时，即可进行回收。

优点：实现简单，垃圾对象便于标识，判定效率高，回收没有延迟性。

缺点：

1. 增加了存储空间的开销。
2. 加法减法，增加了时间开销。
3. 无法处理循环引用的情况。

Python采用了这种算法，Java没有采用

Python采用了weakref解决循环引用。

### 垃圾标记阶段算法-可达性分析算法

又称为根搜索算法、追踪性垃圾收集

“GC Roots”根集合就是一组必须活跃的引用

#### 基本思路

1. 以根对象集合为起始点，按照从上至下的方式 **搜索被根对象集合所连接的目标对象是否可达**。
2. 使用可达性分析算法后，内存中**只有存活的对象**会被根对象集合直接或间接**连接**着，搜索走过的路劲被称为**引用链**
3. 不可达的对象，可以被标记为垃圾对象。

#### GC Roots包含的元素

1. 虚拟机栈中引用的对象
2. 本地方法栈内JNI引用的对象
3. (方法区)类静态属性引用的对象
4. (方法区)常量引用的对象
5. 所有被同步锁synchronized持有的对象
6. Java虚拟机内部的引用（基本数据类型对应的Class对象，类加载器）
7. 反映Java虚拟机内部情况的JMXBean、JVMTi中注册的回调、本地代码缓存
8. 除了固定的GC Roots集合以外，根据用户所选用的垃圾收集器以及当前回收的内存区域不同，**还可以有其他对象“临时性”地加入**，共同构成完整的GC Roots集合，比如：**分代收集和局部回收（Partial GC）**
   * 如果只针对Java堆中的某一块区域进行垃圾回收（比如：典型的只针 对新生代）， 必须考虑到内存区域是虚拟机自己的实现细节，更不是孤立封闭的， 这个区域的对象完全有可能被其他区域的对象所引用，这时候就需要一并将关联的区域对象也加入GC Roots集合中去考虑，才能保证可达性分析的准确性。

由于Root采用栈方式存放变量和指针，如果一个指针，它保存了堆内存里面的对象，但自己又不在堆内存里面，那么它就是一个Root。

#### 注意

* 如果使用可达性分析算法，那么分析工作必须在能保障一致性的快照中进行的。
* 这也是导致GC是必须“Stop The World”的一个重要原因

### 对象finalization的机制

#### 概述

Java语言提供了对象终止（finalization）机制，来允许开发人员**提供对象被销毁之前的自定义处理逻辑。**

对象被回收之前，先调用这个对象的finalize()方法

finalize()方法可被重写，**用于在对象被回收时，进行资源释放**。

#### 不要主动调用finalize的理由

永远不要主动调用finalize()方法，应该交给垃圾回收机制。

1. finalize()时可能会导致对象复活
2. finalize()方法的执行时间是没有保障的，它完全由GC线程决定，极端情况下，若不发生GC，则finalize()方法将没有执行的机会
3. 一个糟糕的finalize()会严重影响GC性能。

从功能上来说，finalize()方法与C++中的析构函数比较相似，但Java采用的是基于垃圾回收器的自动内存管理机制。所以finalize()在本质上不同于C++中的析构函数。

#### 虚拟机中对象可能处于的状态

1. 可触及的：从根节点开始，可以达到这个对象
2. 可复活的：对象的所有引用都被释放，但是对象有可能在finalize()中复活。
3. 不可触及的：对象的finalize()的调用，并且没有复活，那么就会进入不可触及状态。**不可触及状态不可复活**，**因为finalize只能调用一次**。

#### 具体过程

1. 如果对象objA到GC Roots没有引用连，则进行一次标记
2. 判断此对象有没有必要执行finalize方法
   1. 如果没有重写finalize或者执行过一次finzlize方法，则被虚拟机视为“没有必要执行”，objA被判断为不可触及的
   2. 如果对象objA重写了finalize方法，但还未执行过，那么objA会被插入到F-Queue队列中，由一个虚拟机自动创建的、低优先级的Finalizer线程触发器finalize方法的执行。
   3. **finalize方法是对象逃脱死亡的最后机会**，稍后GC会对F-Queue队列中的对象进行第二次标记。**如果objA在finalize()方法中与引用链上的任何一个对象建立的联系**，那么在第二次标记时，objA会被移除“即将回收”集合。之后，如果对象再次出现没有引用存在的情况下，finalize方法不会被调用，对象会直接编程不可触及的状态（**一个对象的finalize方法只会被调用一次**）

