深度解析 Java 内存原型

一、Java 虚拟机内存原型

寄存器：我们在程序中无法控制。
栈：存放基本类型的数据和对象的引用，但对象本身不存放在栈中，而是存放在堆中。
堆：存放用 new 产生的数据。
静态域：存放在对象中用 static 定义的静态成员。
常量池：存放常量。
非 RAM 存储：硬盘等永久存储空间。

二、常量池（constant pool）

常量池指的是在编译期被确定，并被保存在已编译的 class 文件中的一些数据。除了包含代码中所定义的各种基本类型（如 int、long 等等）和对象型（如 String 及数组）的常量值（final）外，还包含一些以文本形式出现的符号引用，比如：
1、类和接口的全限定名；
2、字段的名称和描述符；
3、方法的名称和描述符。

虚拟机必须为每个被装载的类型维护一个常量池。常量池就是该类型所用到常量的一个有序集合，包括直接常量（string,integer 等）和其他类型：字段和方法的符号引用。**对于 String 常量，它的值是在常量池中的。**而 JVM 中的常量池在内存当中是以表的形式存在的，对于 String 类型，有一张固定长度的 CONSTANT_String_info 表用来存储文字字符串值，注意：该表只存储文字字符串值，不存储符号引用。说到这里，对常量池中的字符串值的存储位置应该有一个比较明了的理解了。在程序执行的时候，常量池会储存在 Method Area，而不是堆中。

三、Java 内存分配中的栈

栈的基本单位是帧（或栈帧）：每当一个 Java 线程运行的时候，Java 虚拟机会为该线程分配一个 Java 栈。该线程在执行某个 Java 方法的时候，向 Java 栈压入一个帧，这个帧用于存储参数、局部变量、操作数、中间运算结果等。当这个方法执行完的时候，帧会从栈中弹出。Java 栈上的所有数据是私有的，其他线程都不能访问该线程的栈数据。在函数中定义的一些基本类型的变量数据和对象的引用变量都在函数的栈内存中分配。当在一段代码块中定义一个变量时，Java 就在栈中为这个变量分配内存空间，当该变量退出该作用域后，Java 会自动释放掉为该变量所分配的内存空间，该内存空间可以立即被另作他用。

四、Java 内存分配中的堆

Java 虚拟机中的堆用来存放由 new 创建的对象和数组。在堆中分配的内存，由 Java 虚拟机的自动垃圾回收机制来管理堆的内存。简单的说和栈相对，堆主要是用来存放 Java 对象的，栈主要是用来存放对象引用的。在堆中产生了一个数组或对象后，还可以在栈中定义一个特殊的变量，让栈中这个变量的取值等于数组或对象在堆内存中的首地址，栈中的这个变量就成了数组或对象的引用变量。引用变量就相当于是为数组或对象起的一个名称，以后就可以在程序中使用栈中的引用变量来访问堆中的数组或对象。引用变量就相当于是为数组或对象起的一个名称。

引用变量是普通的变量，定义时在栈中分配，引用变量在程序运行到其作用域之外后被释放。而数组和对象本身在堆中分配，即使程序运行到使用 new 产生数组或者对象的语句所在的代码块之外，数组和对象本身占据的内存不会被释放，数组和对象在没有引用变量指向它的时候，才变为垃圾，不能再被使用，但仍然占据内存空间不放，在随后的一个不确定的时间被垃圾回收器收走（释放掉）。这也是 Java 比较占内存的原因。实际上，栈中的变量指向堆内存中的变量，这就是 Java 中的指针！

java 的堆是一个运行时数据区，类的对象从中分配空间。这些对象通过 new + 构造器等指令建立，它们不需要程序代码来显式的释放。堆是由垃圾回收来负责的，**堆的优势**是可以动态地分配内存大小，生存期也不必事先告诉编译器，因为它是在运行时动态分配内存的，Java 的垃圾收集器会自动收走这些不再使用的数据。但缺点是，由于要在运行时动态分配内存，存取速度较慢。

栈的优势是存取速度比堆要快，仅次于寄存器，栈数据可以共享。但缺点是，存在栈中的数据大小与生存期必须是确定的，缺乏灵活性。栈中主要存放一些基本类型的变量数据（int，short，long，byte，float，double，boolean，char）和对象引用。

栈有一个很重要的特殊性，就是存在栈中的数据可以共享。假设我们同时定义： int a=3；int b=3；编译器先处理 int a = 3；首先它会在栈中创建一个变量为 a 的引用，然后查找栈中是否有 3 这个值，如果没找到，就将 3 存放进来，然后将 a 指向 3，接着处理 int b = 3；在创建完 b 的引用变量后，因为在栈中已经有 3 这个值，便将 b 直接指向 3 这样，就出现了 a 与 b 同时均指向 3 的情况。

这时，如果再令 a=4；那么编译器会重新搜索栈中是否有 4 值，如果没有，则将 4 存放进来，并令 a 指向 4；如果已经有了，则直接将 a 指向这个地址。因此 a 值的改变不会影响到 b 的值。

要注意这种数据的共享与两个对象的引用同时指向一个对象的这种共享是不同的，因为这种情况 a 的修改并不会影响到 b，它是由编译器完成的，它有利于节

省空间。而一个对象引用变量修改了这个对象的内部状态，会影响到另一个对象引用变量。