【 首先，有必要说明一件事情 - GC Root是对象而不是ref 】

问题：GC Root是以下的哪个?

eg：**Obj obj = new Obj();**

答: **Obj()对象;**

解析：

相对于**GC Roots**对象集而言，obj 是一种身份的**证明;**

【换言之，通常来说】：

创建一个对象，对象会在堆上**【开辟一块空间】**，同时会将这块**【所开辟空间的地址】**作为**【引用】**

----------**读懂上述再往下看**----------

Q：Obj obj = new Obj()：obj 存放在哪儿?

A：不确定，看看作用域是在哪儿;

若是方法中调用，则obj在栈中;

若是类的属性，则保存在方法区【理论上方法区也是堆】中;

问题：可作为GC Roots的对象有哪几种?

答:

①虚拟机栈（局部变量表）引用的对象;

②方法区中（类静态属性）引用的对象；eg:

③方法区中（常量）引用的对象;

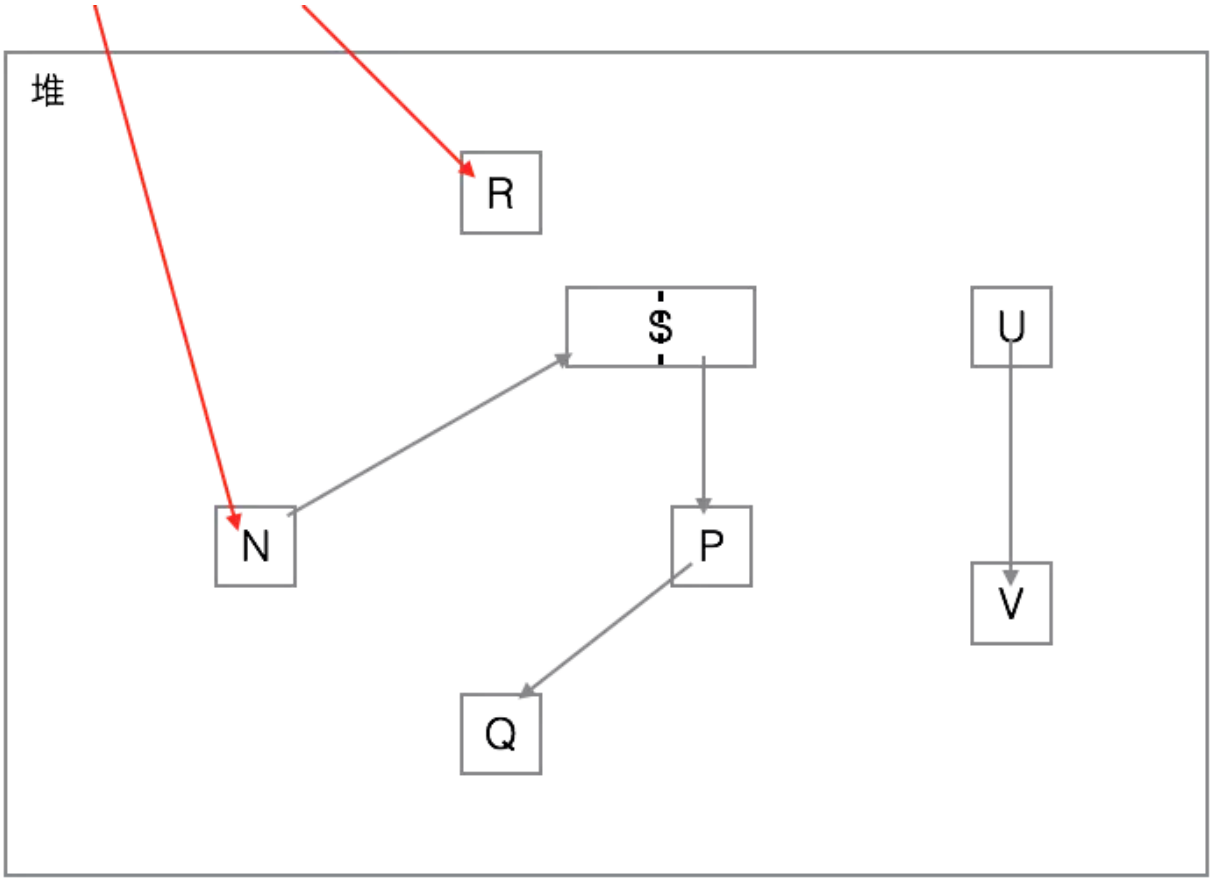
④本地方法栈中JNI引用的对象;

总结:

栈（虚拟机栈、本地方法栈），**方法区**中的 **static ref** 可以**作为**GC Root;

GC Root作为上位者，可以主动切断与堆中对象的联系 （ eg： obj = null），

使得**堆中相应的对象**在**引用链（ref chain）**上不可达，进而被判断为垃圾;



gc root

堆是被我们垃圾回收所管理的内存空间。

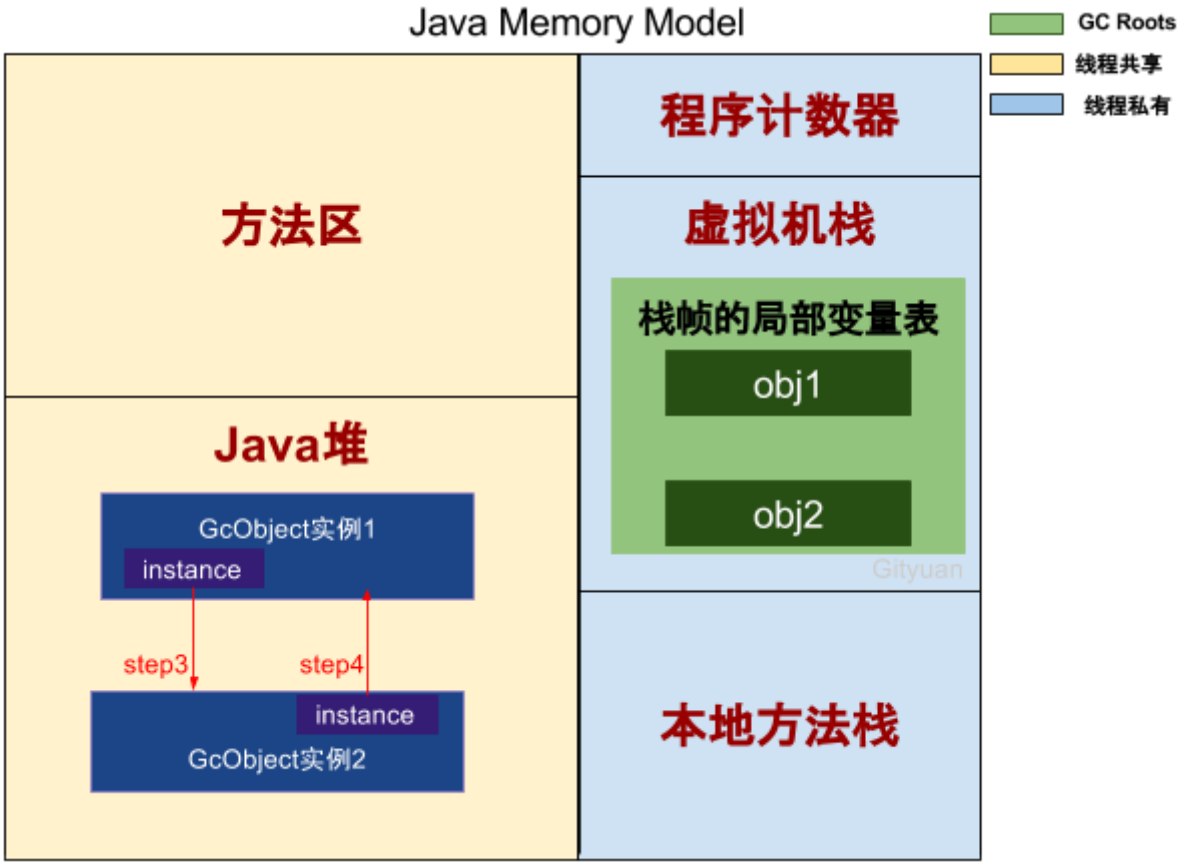
如图，存在两种引用，一种是堆外对象对堆内对象的引用，被标注为红色；另外一种是堆内对象之间的引用，被标注为灰色。

通常我们说的gc root就可以被认为是红色的那种引用，比如说栈引用堆中对象。

为什么我们不认为堆内对象之间的引用是gc root呢？**因为我们的对象，最终是要被外部使用的，比如说被栈引用所访问。**

因此，如果一大堆的堆内对象之间互相引用，但是没有任何堆外部引用，那么这部分对象实际上也是不可达的。

HotSpot就是如此的，所有的堆中的对象，最终都是被栈所使用的。因而，U和V就可以看做是不可达的对象了。



(1)首先第一种是【虚拟机栈】中的引用的对象，我们在程序中**正常创建一个对象**，对象会在堆上**【开辟一块空间】**，**同时会将这块【空间的地址】作为【引用】**保存到虚拟机栈中，如果对象生命周期结束了，那么引用就会从虚拟机栈中出栈，因此如果在虚拟机栈中有引用，就说明这个对象还是有用的，这种情况是最常见的。

(2)第二种是我们在类中定义了全局的静态的对象，也就是使用了【**static**】关键字，由于虚拟机栈是**线程私有**的，所以这种对象的引用会保存在共有的方法区中，**显然将方法区中的静态引用作为GC Roots是必须的。**

(3)第三种便是常量引用，就是使用了【**final**】关键字，由于这种引用初始化之后不会修改，所以方法区常量池里的引用的对象也应该作为GC Roots。

(4)最后一种是在使用【JNI】技术时，有时候单纯的Java代码并不能满足我们的需求，我们可能需要在Java中调用C或C++的代码，因此会使用native方法，JVM内存中专门有一块本地方法栈，用来保存这些对象的引用，所以本地方法栈中引用的对象也会被作为GC Roots。