**Shallow Size、Retained Size、Heap Size和Allocated**

**Shallow Size：**

对象本身占用的内存空间，不包含其引用的对象，但在JAVA中除基本类型外，一切均为对象，也就是说持有的一直为对象的引用，如String类型对象，它主要包含3个int成员（3\*4B)、1个char[]成员(1\*4B)以及一个对象头(8B),尽管char[]可能指向一大块字符，但String对象里只有一个引用所占4B的空间，因此String类型对象的Shallow Size就是12B+4B+8B = 24B。

**Retained Size:**

对象本身的Shallow Size + 对象能直接或间接访问到的对象的Shallow Size，也就是说Retained Size就是该对象被[gc](https://www.baidu.com/s?wd=gc&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "_blank)之后所能回收内存的总和。

为了更好地理解Retained Size，不妨以图1加以说明，图中每个节点对应内存中的一个对象，但注意这里有个特殊的节点GC Roots(详见注解），它也是Reference chain的起点。在图3中，从对象obj1入手，可知通过obj1可以直接或间接访问的对象为蓝色节点，而左图中obj3是GC Roots[可达](https://www.baidu.com/s?wd=%E5%8F%AF%E8%BE%BE&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)的，也就是说断掉obj1到obj3的Reference chain后，obj3还有GC Roots的Reference chain，因而在gc时obj1被回收也不会回收obj3，因此在左图obj1的obj1的Retained Size = obj1 + obj2 + obj4 的shallow size，而右图obj1的Retained Size = obj1 + obj2 + obj4 +obj3 的shallow size.总之，Retained size是一个整体度量，能反映内存结构和对象图的依赖关系，还可以找到根节点。

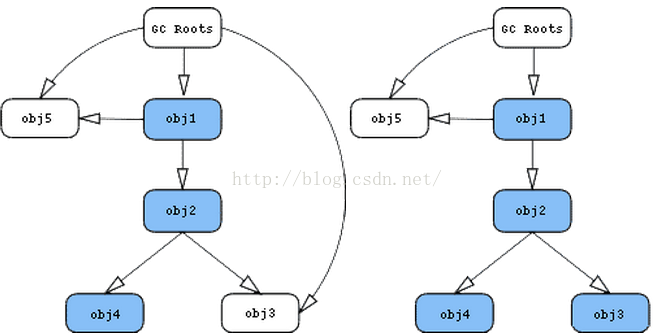


图1 Retained Size示例图

注：GC Roots特指垃圾收集器对象，JVM GC时会回收那些不是GC roots且没有被GC roots引用的对象。JVM对内存中对象进行回收，主要是通过GC Roots Tracing来辨别对象是否在使用，进而进行回收。也就是说通过一系列名为“GC Roots”的对象作为起始点，从节点向下搜索，搜索过的路径称之为Reference Chain，当一个对象到GC Roots没有任何Reference Chain相连时，则表明此对象不可用，GC时将被回收（该流程的示例图见图2）。

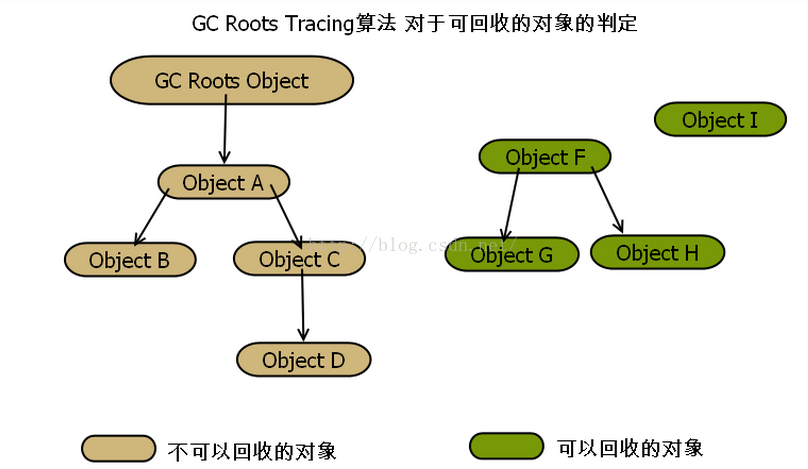


图2.GC Roots Tracing回收机制

**Heap Size:**

堆的大小，当资源增加，当前堆的空间不够时，系统会增加堆的大小，若超过上限（如64M，阈值视平台而定）则会被杀掉

**Allocated:**

堆中已分配的大小，即App应用实际占用的内存大小，资源回收后，此项数据会变小。

**建议:**若单一操作反复进行，堆大小一直增加，则有内存泄露的隐患，可采用MAT进一步查看。