**java的四种引用类型：强、弱、软、虚（就是对象被引用的时候，这四种类型对应被回收的时机。而我们平时的可达性分析，其实经常讲的是强引用，因为对象可达就不会被回收了，但是这儿就有可能会，比如弱，虚，就每次GC就回收掉了。而软则等到内存满的时候会回收掉。软，弱，用于缓存，虚用于跟踪对象的状态。）**

1、强引用（StrongReference）

强引用是使用最普遍的引用。如果一个对象具有强引用，那垃圾回收器绝不会回收它。如下：

Object o=new Object(); // 强引用

当内存空间不足，Java虚拟机宁愿抛出OutOfMemoryError错误，使程序异常终止，**也不会靠随意回收具有强引用的对象来解决内存不足的问题。**如果不使用时，要**通过如下方式来弱化引用**，如下：

**o=null; // 帮助垃圾收集器回收此对象   
显式地设置o为null**，或超出对象的生命周期范围，则gc认为该对象不存在引用，这时就可以回收这个对象。具体什么时候收集这要取决于gc的算法。

举例：

public void test(){

Object o=new Object();

// 省略其他操作

}

在一个方法的内部有一个强引用，这个引用保存在栈中，而真正的引用内容（Object）保存在堆中。**当这个方法运行完成后就会退出方法栈，则引用内容的引用不存在，这个Object会被回收。**

但是如果这个**o是全局的变量时**，就需要在**不用这个对象时赋值为null，因为强引用不会被垃圾回收。**  
强引用在实际中有非常重要的用处，举个ArrayList的实现源代码：

private transient Object[] elementData;

public void clear() {

modCount++;

// Let gc do its work

for (int i = 0; i < size; i++)

elementData[i] = null;

size = 0;

}

在ArrayList类中定义了一个私有的变量elementData数组，**在调用方法清空数组时可以看到为每个数组内容赋值为null。**不同于**elementData=null，强引用仍然存在，**避免在后续调用 add()等方法添加元素时进行重新的内存分配。使用如clear()方法中释放内存的方法对数组中存放的引用类型特别适用，这样就可以及时释放内存。

2、软引用（SoftReference）

如果一个对象只具有软引用，则内存空间足够，垃圾回收器就不会回收它；如果内存空间不足了，就会回收这些对象的内存。**只要垃圾回收器没有回收它，该对象就可以被程序使用。软引用可用来实现内存敏感的高速缓存。(软引用就是本来已经存在的对象（比如已经被引用的状态了），但是在内存吃紧的时候会回收掉，否则不会回收)**

String str=new String("abc"); // 强引用

**SoftReference**<String> softRef=new SoftReference<String>(str); // 软引用

当内存不足时，等价于：

If(JVM.内存不足()) {

str = null; // 转换为软引用

System.gc(); // 垃圾回收器进行回收

}

虚引用在实际中有重要的应用，例如浏览器的后退按钮。按后退时，这个后退时显示的网页内容是重新进行请求还是从缓存中取出呢？这就要看具体的实现策略了。   
（1）如果一个网页在浏览结束时就进行内容的回收，则按后退查看前面浏览过的页面时，需要重新构建   
（2）如果将浏览过的网页存储到内存中会造成内存的大量浪费，甚至会造成内存溢出   
这时候就可以使用软引用

Browser prev = new Browser(); // 获取页面进行浏览

SoftReference sr = new SoftReference(prev); // 浏览完毕后置为软引用

if(sr.get()!=null){

rev = (Browser) sr.get(); // 还没有被回收器回收，直接获取

}else{

prev = new Browser(); // 由于内存吃紧，所以对软引用的对象回收了

sr = new SoftReference(prev); // 重新构建

}

这样就很好的解决了实际的问题。

**软引用可以和一个引用队列（ReferenceQueue）联合使用，如果软引用所引用的对象被垃圾回收器回收，Java虚拟机就会把这个软引用加入到与之关联的引用队列中。**3、弱引用（WeakReference）**（每次回收必回收掉的对象）**

弱引用与软引用的区别在于：只具有弱引用的对象拥有更短暂的生命周期。在垃圾回收器线程扫描它所管辖的内存区域的过程中，一旦发现了只具有弱引用的对象，不管当前内存空间足够与否，都会回收它的内存。**不过，由于垃圾回收器是一个优先级很低的线程，因此不一定会很快发现那些只具有弱引用的对象。**

String str=new String("abc");

**WeakReference<String> abcWeakRef = new WeakReference<String>(str);**

str=null;

当垃圾回收器进行扫描回收时等价于：

str = null;

System.gc();

如果这个对象是偶尔的使用，并且希望在使用时随时就能获取到，但又不想影响此对象的垃圾收集，那么你应该用 Weak Reference 来记住此对象。   
下面的代码会让**str再次变为一个强引用：**

String abc = abcWeakRef.get();

**弱引用可以和一个引用队列（ReferenceQueue）联合使用，如果弱引用所引用的对象被垃圾回收，Java虚拟机就会把这个弱引用加入到与之关联的引用队列中。**当你想引用一个对象，但是这个对象有自己的生命周期，你不想介入这个对象的生命周期，这时候你就是用弱引用。

这个引用不会在对象的垃圾回收判断中产生任何附加的影响。

public class ReferenceTest {

private static ReferenceQueue<VeryBig> **rq** = new ReferenceQueue<VeryBig>();

public static void checkQueue() {

Reference<? extends VeryBig> ref = null;

**while ((ref = rq.poll()) != null) {**

**if (ref != null) {**

**System.out.println("In queue: " + ((VeryBigWeakReference) (ref)).id);**

}

}

}

public static void main(String args[]) {

int size = 3;

LinkedList<WeakReference<VeryBig>> weakList = new LinkedList<WeakReference<VeryBig>>();

for (int i = 0; i < size; i++) {

weakList.add(new VeryBigWeakReference(new VeryBig("Weak " + i), rq));

System.out.println("Just created weak: " + weakList.getLast());

}

System.gc();

try { // 下面休息几分钟，让上面的垃圾回收线程运行完成

Thread.currentThread().sleep(6000);

} catch (InterruptedException e) {

e.printStackTrace();

}

checkQueue();

}

}

class VeryBig {

public String id;

// 占用空间,让线程进行回收

byte[] b = new byte[2 \* 1024];

public VeryBig(String id) {

this.id = id;

}

protected void finalize() {

System.out.println("Finalizing VeryBig " + id);

}

}

class VeryBigWeakReference extends WeakReference<VeryBig> {

public String id;

public VeryBigWeakReference(VeryBig big, ReferenceQueue<VeryBig> rq) {

super(big, rq);

this.id = big.id;

}

protected void finalize() {

System.out.println("Finalizing VeryBigWeakReference " + id);

}

}

最后的输出结果为：

Just created weak: com.javabase.reference.VeryBigWeakReference@1641c0

Just created weak: com.javabase.reference.VeryBigWeakReference@136ab79

Just created weak: com.javabase.reference.VeryBigWeakReference@33c1aa

Finalizing VeryBig Weak 2

Finalizing VeryBig Weak 1

Finalizing VeryBig Weak 0

In queue: Weak 1

In queue: Weak 2

In queue: Weak 0

4、虚引用（PhantomReference）

“虚引用”顾名思义，就是形同虚设，与其他几种引用都不同，**虚引用并不会决定对象的生命周期。如果一个对象仅持有虚引用，那么它就和没有任何引用一样，在任何时候都可能被垃圾回收器回收。**

虚引用主要**用来跟踪对象被垃圾回收器回收的活动**。虚引用与软引用和弱引用的一个区别在于：**虚引用必须和引用队列 （ReferenceQueue）联合使用。当垃圾回收器准备回收一个对象时，如果发现它还有虚引用，就会在回收对象的内存之前，把这个虚引用加入到与之关联的引用队列中。**

5、总结   
Java4种引用的级别由高到低依次为：

强引用 > 软引用 > 弱引用 > 虚引用

强引用是我们在编程过程中使用的最简单的引用，如代码String s=”abc”中变量s就是字符串对象”abc”的一个强引用。任何被强引用指向的对象都不能被垃圾回收器回收，这些对象都是在程序中需要的。弱引用使用java.lang.ref.WeakReference class 类来表示，你可以使用如下代码创建弱引用：

复制代码 代码如下:

Counter counter = new Counter(); // strong reference - line 1  
WeakReference<Counter> weakCounter = new WeakReference<Counter>(counter); //weak reference  
**counter = null; // now Counter object is eligible for garbage collection**

现在只要你给强引用对象counter赋空值null,该对象就可以被垃圾回收器回收。因为该对象此时不再含有其他强引用，即使指向该对象的**弱引用weakCounter也无法阻止垃圾回收器对该对象的回收**。相反的，**如果该对象含有软引用，Counter对象不会立即被回收，**除非JVM需要内存。Java中的软引用使用java.lang.ref.SoftReference类来表示，你可以使用如下代码创建软引用：

复制代码 代码如下:

Counter prime = new Counter(); // prime holds a strong reference – line 2  
SoftReference soft = new SoftReference(prime) ; //soft reference variable has SoftReference to Counter Object created at line 2  
  
prime = null; // now Counter object is eligible for garbage collection but only be collected when JVM absolutely needs memory

强引用置空之后，代码的第二行为对象Counter创建了一个软引用**，该引用同样不能阻止垃圾回收器回收对象，但是可以延迟回收，**与弱引用中急切回收对象不同。鉴于软引用和弱引用的这一区别，**软引用更适用于缓存机制，而弱引用更适用于存贮元数据**。另一个使用弱引用的例子是WeakHashMap，它是除HashMap和TreeMap之外，Map接口的另一种实现。WeakHashMap有一个特点：map中的键值(keys)都被封装成弱引用，也就是说一旦强引用被删除，WeakHashMap内部的弱引用就无法阻止该对象被垃圾回收器回收。

虚引用是java.lang.ref package包中第三种可用的引用，使用java.lang.ref.PhantomReference类来表示。**拥有虚引用的对象可以在任何时候被垃圾回收器回收。**和弱引用和软引用相似，你可以通过如下代码创建虚引用：

复制代码 代码如下:

DigitalCounter digit = new DigitalCounter(); // digit reference variable has strong reference – line 3  
PhantomReference phantom = new PhantomReference(digit); // phantom reference to object created at line 3  
 **digit = null;**

一旦移除强引用，第三行的DigitalCounter对象可以在任何时候被垃圾回收器回收。因为只有一个虚引用指向该对象，而虚引用无法阻止垃圾回收器回收对象。

除了了解弱引用、软引用、虚引用和WeakHashMap，还需要了解ReferenceQueue。在创建任何弱引用、软引用和虚引用的过程中你可以通过如下代码**提供引用队列ReferenceQueue：**

复制代码 代码如下:

ReferenceQueue refQueue = new ReferenceQueue(); //reference will be stored in this queue for cleanup  
DigitalCounter digit = new DigitalCounter();  
PhantomReference<DigitalCounter> phantom = new PhantomReference<DigitalCounter>(digit, refQueue);



# 5 引用队列（ReferenceQueue）

效果：引用队列可以配合软引用、弱引用及幽灵引用使用，当引用的对象将要被JVM回收时，会将其加入到引用队列中。

应用：通过引用队列可以了解JVM垃圾回收情况。