**Java强引用、软引用、弱引用、虚引用(幻象引用)详解**

<https://blog.csdn.net/hnhygkx/article/details/80182906>

## 一、概述：

众所周知，Java中是JVM负责内存的分配和回收，这是它的优点（使用方便，程序不用再像使用c那样操心内存），但同时也是它的缺点（不够灵活）。为了解决内存操作不灵活这个问题，可以采用软引用等方法。   
在JDK1.2以前的版本中，当一个对象不被任何变量引用，那么程序就无法再使用这个对象。也就是说，只有对象处于可触及状态，程序才能使用它。这 就像在日常生活中，从商店购买了某样物品后，如果有用，就一直保留它，否则就把它扔到垃圾箱，由清洁工人收走。一般说来，如果物品已经被扔到垃圾箱，想再 把它捡回来使用就不可能了。   
但有时候情况并不这么简单，你可能会遇到类似鸡肋一样的物品，食之无味，弃之可惜。这种物品现在已经无用了，保留它会占空间，但是立刻扔掉它也不划算，因 为也许将来还会派用场。对于这样的可有可无的物品，一种折衷的处理办法是：如果家里空间足够，就先把它保留在家里，如果家里空间不够，即使把家里所有的垃 圾清除，还是无法容纳那些必不可少的生活用品，那么再扔掉这些可有可无的物品。   
从JDK1.2版本开始，把对象的引用分为四种级别，从而使程序能更加灵活的控制对象的生命周期。这四种级别由高到低依次为：强引用、软引用、弱引用和虚引用。

## 二、具体描述：

**1．强引用**   
以前我们使用的大部分引用实际上都是强引用，这是使用最普遍的引用。如果一个对象具有强引用，那就类似于必不可少的生活用品，垃圾回收器绝不会回收它。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问题。   
如

1. String str = "abc";
2. List<String> list = new Arraylist<String>();
3. list.add(str);

* 1
* 2
* 3

在list集合里的数据不会释放，即使内存不足也不会

* 1
* 2

**2、软引用（SoftReference）**   
如果一个对象只具有软引用，那就类似于可有可物的生活用品。如果内存空间足够，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，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。   
如：

1. public class Test {
3. public static void main(String[] args){
4. System.out.println("开始");
5. A a = new A();
6. SoftReference<A> sr = new SoftReference<A>(a);
7. a = null;
8. if(sr!=null){
9. a = sr.get();
10. }
11. else{
12. a = new A();
13. sr = new SoftReference<A>(a);
14. }
15. System.out.println("结束");
16. }
18. }
20. class A{
21. int[] a ;
22. public A(){
23. a = new int[100000000];
24. }
25. }

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25

当内存足够大时可以把数组存入软引用，取数据时就可从内存里取数据，提高运行效率

**3．弱引用（WeakReference）**   
如果一个对象只具有弱引用，那就类似于可有可物的生活用品。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。   
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。   
如：

1. Object c = new Car(); *//只要c还指向car object, car object就不会被回收*
2. WeakReference<Car> weakCar = new WeakReference(Car)(car);

* 1
* 2

当要获得weak reference引用的object时, 首先需要判断它是否已经被回收:

weakCar.get();

* 1

如果此方法为空, 那么说明weakCar指向的对象已经被回收了.

下面来看一个例子:

1. public class Car {
2. private double price;
3. private String colour;
5. public Car(double price, String colour){
6. this.price = price;
7. this.colour = colour;
8. }
10. public double getPrice() {
11. return price;
12. }
13. public void setPrice(double price) {
14. this.price = price;
15. }
16. public String getColour() {
17. return colour;
18. }
19. public void setColour(String colour) {
20. this.colour = colour;
21. }
23. public String toString(){
24. return colour +"car costs $"+price;
25. }
27. }

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27

1. public class TestWeakReference {

4. public static void main(String[] args) {
6. Car car = new Car(22000,"silver");
7. WeakReference<Car> weakCar = new WeakReference<Car>(car);
9. int i=0;
11. while(true){
12. if(weakCar.get()!=null){
13. i++;
14. System.out.println("Object is alive for "+i+" loops - "+weakCar);
15. }else{
16. System.out.println("Object has been collected.");
17. break;
18. }
19. }
20. }
22. }

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22

在上例中, 程序运行一段时间后, 程序打印出”Object has been collected.” 说明, weak reference指向的对象的被回收了.

如果要想打出的是   
Object is alive for “+i+” loops - “+weakCar

那么只要在这句话前面加上   
System.out.println(“car==== “+car);   
因为在此强引用了car对象

**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。   
虚引用主要用来跟踪对象被垃圾回收的活动。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。程序可以通过判断引用队列中是 否已经加入了虚引用，来了解   
被引用的对象是否将要被垃圾回收。程序如果发现某个虚引用已经被加入到引用队列，那么就可以在所引用的对象的内存被回收之前采取必要的行动。   
特别注意，在世纪程序设计中一般很少使用弱引用与虚引用，使用软用的情况较多，这是因为软引用可以加速JVM对垃圾内存的回收速度，可以维护系统的运行安全，防止内存溢出（OutOfMemory）等问题的产生。

## 总结：

强引用：   
String str = “abc”;   
list.add(str);   
软引用：   
如果弱引用对象回收完之后，内存还是报警，继续回收软引用对象   
弱引用：   
如果虚引用对象回收完之后，内存还是报警，继续回收弱引用对象   
虚引用：   
虚拟机的内存不够使用，开始报警，这时候垃圾回收机制开始执行System.gc(); String s = “abc”;如果没有对象回收了， 就回收没虚引用的对象