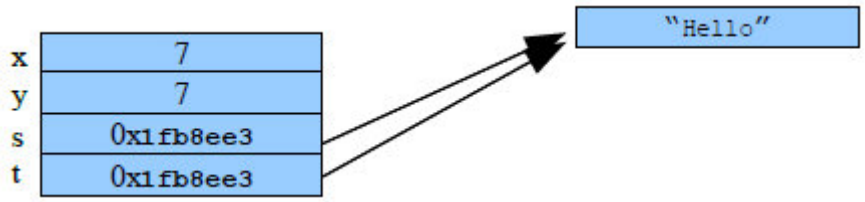
引用类型

- **对象、数组都是引用数据类型，指向对象的变量是引用变量。除基本数据类型都是引用数据类型。**
- 引用型变量能被重置为其他对象，除了声明为**final**的对象。
- 所有引用类型的默认值都是**null**。
- 一个引用变量可以用来引用与任何与之兼容的类型。可以声明为**类类型**或者**接口类型**。

引用类型的赋值：

```
int x = 7;
int y = x;
String s = "Hello";
String t = s;
```



常量

若内存地址不变，则值也不可以改变的东西称为常量。用final标志，声明方式和变量类似：

```
final double PI = 3.1415927;
```

byte、int、long、和short都可以用十进制、16进制以及8进制的方式来表示。Java的字符串常量也是包含在**两个引号**之间的字符序列。例如：

```
char a = '\u0001';
String a = "\u0001";
```

静态常量与常量

static+final

静态常量，编译期常量，编译时就确定值存入类的常量池。放于方法区中的静态常量池。

如果调用此常量的类不是定义常量的类，那么不会初始化定义常量的类，因为在编译阶段通过常量传播优化，已经将常量存到调用类的常量池中了

```
class ConstC{
        static{
                System.out.println("ConstC init!");
        }
        public ConstC(){
                System.out.println("ConstC ");
        }
        public static final String HELLO = "hello world!";
}
public class NotInit {

        public static void main(String[] args) {
                //经过编译优化，静态常量HELLO已经存到NotInit类自身常量池中，不会加载ConstC
                System.out.println(ConstC.HELLO);
        }

}
//输出:hello world!
```

final

常量，类加载时确定或者更靠后。

当用final作用于类的成员变量时，成员变量（注意是类的成员变量，局部变量只需要保证在使用之前被初始化赋值即可）必须在定义时或者构造器中进行初始化赋值

对于一个final变量，如果是基本数据类型的变量，则其数值一旦在初始化之后便不能更改；

如果是引用类型的变量，则在对其初始化之后便不能再让其指向另一个对象。但是它指向的对象的内容是可变的

Types of References in Java

引用的级别由高到低依次为：

强引用 > 软引用 > 弱引用 > 虚引用

- Strong References

引用对象默认的类型。当指向Null的时候才回收。

```
MyClass obj = new MyClass ();
obj = null;
     //'obj' object is no longer referencing to the instance.
     So the 'MyClass type object is now available for garbage collection.
```

- Weak References

使用时需明确指定。gc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。但较慢才能发现。

  - In WeakHashMap to reference the entry objects .

- An object with only weak references （i.e. no strong or soft references linked to any object）,this object will be marked for garbage collection.
- To create references java.lang.ref.WeakReference class is used.

```
	String str=new String("abc");
	WeakReference<String> abcWeakRef = new
WeakReference<String>(str);
	str=null;
```

对象偶尔使用，随时能获取，不影响垃圾收集。又会变为Strong References。

```
	String  abc = abcWeakRef.get();
```

- Soft References

若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。

```
	String str=new String("abc");
// 强引用
	SoftReference<String> softRef=new SoftReference<String>
(str);
```

- Phantom References

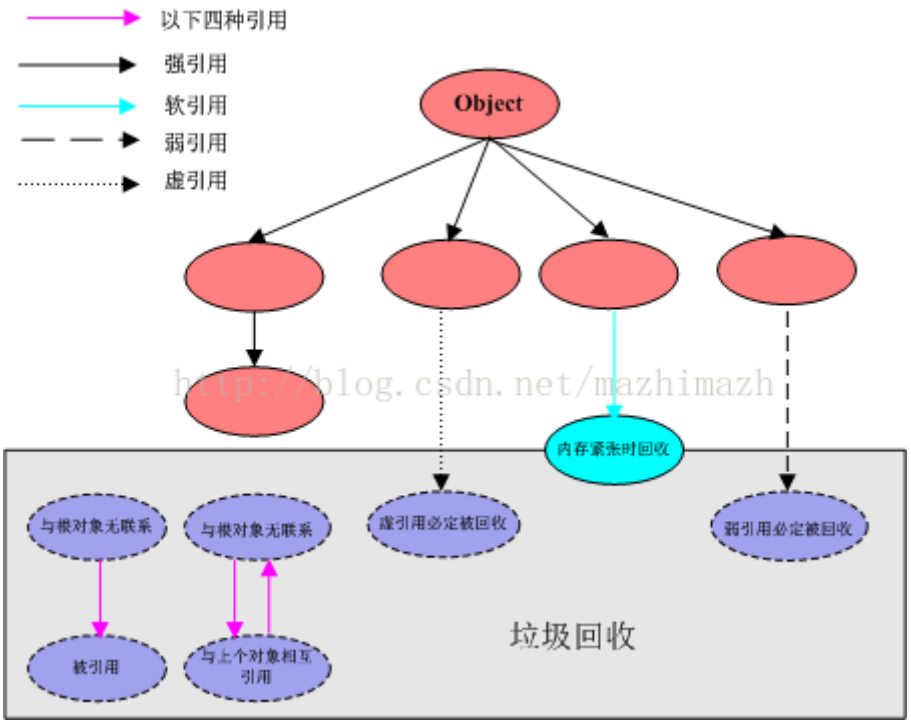
如果一个对象仅持有虚引用，那么它就和没有任何引用一样，在任何时候都可能被垃圾回收器回收。虚引用主要用来跟踪对象被垃圾回收器回收的活动。虚引用与软引用和弱引用的一个区别在于：虚引用必须和引用队列 （ReferenceQueue）联合使用。当垃圾回收器准备回收一个对象时，如果发现它还有虚引用，就会在回收对象的内存之前，把这个虚引用加入到与之 关联的引用队列中。

```
Gfg g = new Gfg();
g.x();

ReferenceQueue<Gfg> refQueue = new ReferenceQueue<Gfg>();

PhantomReference<Gfg> phantomRef = null;
phantomRef = new PhantomReference<Gfg>(g,refQueue);

g = null;
g = phantomRef.get();
```



引用类型	被垃圾回收时间	用途	生存时间
强引用	从来不会	对象的一般状态	JVM停止运行时终止
软引用	在内存不足时	对象缓存	内存不足时终止

弱引用	在垃圾回收时	对象缓存	gc运行后终止
虚引用	Unknown	Unknown	Unknown