Java 中强应用和弱引用

一、概述

众所周知，**Java** 中是由 JVM 负责内存的分配和回收，它的优点是使用方便，程序员不用再像使用 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，可以采用弱引用等方法。

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。

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 版本开始便有了对象的引用的机制，从而使程序能更加灵活的控制对象的生命周期。以下我们就详细说明强引用和弱引用。

二、具体描述

强引用（StrongReference）

它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。强引用是使用最普遍的引用，以前我们使用的大部分引用实际上都是强引用。例如：

```
Student student = new Student();
```

只要此引用存在没要被释放(没有使 student =null)，垃圾回收器永远不会回收。只有当这个引用被释放之后，垃圾回收器才可能回收，这也是我们经常所用到的编码形式。

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。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， 因此不一定会马上发现那些只具有弱引用的对象）如：

```
Student student = new Student(); //只要student还指向Student就不会被回收
WeakReference<Student> weakStudent = new WeakReference<Student>( student);
```

当要获得 weak reference 引用的 student 时,可以使用:

```
weakStudent.get();
```

如果此方法返回的为空，那么说明 weakStudent 指向的对象 student 已经被回收了。

例如我们常用的在内部类的 Handler 中使用的 Activity 弱引用，防止内存泄漏：

```java
public class MainActivity extends Activity {

    private Handler handler =null;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        handler=new MyHandler (this);
    }
    private static class MyHandler extends Handler {
        WeakReference<MainActivity> weakReference ;
        public MyHandler(MainActivity activity ){
            weakReference = new WeakReference<MainActivity>( activity) ;
        }
        public void handleMessage(Message msg) {
            super.handleMessage(msg);
            if ( weakReference.get() != null ){
                //更新ui
            }
        }
    }
}
```