Java 值传递与引用传递的一些误区

作者：catty
原文链接：https://ld246.com/article/1497619255472
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

关于Java值传递与引用传递的讨论有很多，先来了解一下值传递和引用传递。

值传递：是指在调用函数时将实际参数复制一份传递到函数中，这样在函数中如果对参数进行修改，不会影响到实际参数。

引用传递：是指在调用函数时将实际参数的地址传递到函数中，那么在函数中对参数所进行的修改，影响到实际参数。

概念知道了，接下来看问题：

```
public static void main(String[] args) {
    int a = 0;
    char[] c = {'a', 'b', 'c'};
    change(a, c);
    System.out.println("a:" + a);
    System.out.println("c[0]:" + c[0]);
}

private static void change(int a, char[] c) {
    a = 2;
    c[0] = '0';
}
```

输出结果为：

```
a:0
c[0]:0
```

这种写法很容易让人产生误解，认为change方法里直接对a重新赋值，直接对c[0]重新赋值，我们换写法：

```
public static void main(String[] args) {
    int a = 0;
    char[] c = {'a', 'b', 'c'};
    change(a, c);
    System.out.println("a:" + a);
    System.out.println("c[0]:" + c[0]);
}

/**
 * 为避免混淆，换个参数名
 */
private static void change(int aa, char[] cc) {
    aa = 2;
    cc[0] = '0';
}
```

输出结果为：

```
a:0
c[0]:0
```

change方法里对aa的修改并没有影响到main方法里a的值，所以第一个参数是值传递没有问题。但对cc[0]的修改却影响到了main方法里c[0]的值，这是否说明第二个参数是引用传递呢？我们再来修一下代码：

```
public static void main(String[] args) {
    int a = 0;
    char[] c = {'a', 'b', 'c'};
    change(a, c);
    System.out.println("a:" + a);
    System.out.println("c[0]:" + c[0]);
}

/**
 * 为避免混淆，换个参数名
 */
private static void change(int aa, char[] cc) {
    aa = 2;
    cc = null;
}
```

输出结果为：

```
a:0
c[0]:a
```

由此可知，对cc的修改其实并不会影响main方法里的c。那么刚才为什么可以影响呢？因为调用change方法时，我们把c当做参数传了进去，而在change方法里，又有一个形参cc，此时这两个引用同时向同一个内存地址，也就是说，这时候不管用哪个引用去操作这块内存区域的数组，都会导致数组发变化，但是请注意，这里说的是对数组进行操作，如果在change方法里把cc赋值为null，这个就不是数组进行操作了，而是把cc由指向数组变为指向null，相当于有两个变量，一个是c，一个是cc，不管cc赋值成什么，都与c无关。所以，这里并没有引用传递，也是值传递。

再来看一个进阶版：

```
public static void main(String[] args) {
    String str = "abc";
    char[] c = {'a', 'b', 'c'};
    change(str, c);
    System.out.println("str:" + str);
    System.out.println("c[0]:" + c[0]);
}

/**
 * 为避免混淆，换个参数名
 */
private static void change(String s, char[] cc) {
    s = "xyz";
    cc[0] = '0';
}
```

输出结果为：

```
str:abc
c[0]:0
```

两个参数都是引用数据类型，怎么会一个改变了，一个没有改变呢？有些解释是，string是final修饰，不能改变，所以虽然string是引用数据类型，但是它是个特例，是值传递，但是数组能改变，是引传递。通过上面的分析，我们已经知道，不管是引用数据类型，还是基本数据类型，其参数传递都是传递，但是引用数据类型是可以改变其属性的，那为什么string没有改变，难道真的是特例吗？

我们都知道，字符串其实也就相当于是复杂点的char数组，仔细观察会发现，change方法里面对字串s的操作其实是相当于改变其引用，并没有对字符串里的数组进行属性改变，而对数组cc的操作却改变数组的属性，也就是说，这两个操作根本不在一个层面。要么就都改变引用，要么就都改变属性操作保证一致，才能够说明问题。

修改代码，使change改变属性：

```
public static void main(String[] args) throws Exception {
    String str = "abc";
    char[] c = {'a', 'b', 'c'};
    change(str, c);
    System.out.println("str:" + str);
    System.out.println("c[0]:" + c[0]);
}

/**
 * 为避免混淆，换个参数名
 */
private static void change(String s, char[] cc) throws Exception {
    Field valueFieldOfString = String.class.getDeclaredField("value");
    valueFieldOfString.setAccessible(true);
    char[] value = (char[]) valueFieldOfString.get(s);
    value[0] = '0';
    cc[0] = '0';
}
```

输出结果为：

```
str:0bc
c[0]:0
```

通过反射获取到string的私有char数组属性，并修改数组的第一个元素，说明string并不是不能修改也不是什么特例。change方法影响到了main方法里的原始变量，但是原因是我们上面分析的，只是为两个引用都指向同一个内存地址而已，本质是值传递，并不是引用传递。

修改代码，使change改变引用：

```
public static void main(String[] args) {
    String str = "abc";
    char[] c = {'a', 'b', 'c'};
    change(str, c);
    System.out.println("str:" + str);
    System.out.println("c[0]:" + c[0]);
}
```

```
/**
 * 为避免混淆，换个参数名
 */
private static void change(String s, char[] cc) {
    s = null;
    cc = null;
}
```

输出结果为：

```
str:abc
c[0]:a
```

并没有影响main方法中原始变量，值传递。

通过上面的一系列分析，我们可以确定，Java中不管是基本数据类型，还是引用数据类型，都只存在传递，不存在引用传递，更不存在什么特例。