

以下内容节选自 Java 私塾自编教材

### 按值传递还是按引用传递

---

这个在 Java 里面是经常被提起的问题，也有一些争论，似乎最后还有一个所谓的结论：“在 Java 里面参数传递都是按值传递”。事实上，这很容易让人迷惑，下面先分别看看什么是按值传递，什么是按引用传递，只要能正确理解，至于称作按什么传递就不是个大问题了。

#### 3.1: 按值传递是什么

指的是在方法调用时，传递的参数是按值的拷贝传递。示例如下：

```
public class TempTest {
    private void test1(int a){
        //做点事情
    }

    public static void main(String[] args) {
        TempTest t = new TempTest();
        int a = 3;
        t.test1(a); //这里传递的参数a就是按值传递
    }
}
```

按值传递重要特点：传递的是值的拷贝，也就是说传递后就互不相关了。

示例如下：

```
public class TempTest {
    private void test1(int a){
        a = 5;
        System.out.println("test1方法中的a==="+a);
    }

    public static void main(String[] args) {
        TempTest t = new TempTest();
        int a = 3;
        t.test1(a); //传递后，test1方法对变量值的改变不影响这里的a
        System.out.println("main方法中的a==="+a);
    }
}
```

运行结果是：

```
test1方法中的a===5
main 方法中的 a===3
```

#### 3.2: 按引用传递是什么

指的是在方法调用时，传递的参数是按引用进行传递，其实传递的引用的地址，也就是变量所对应的内存空间的地址。

示例如下：

```
public class TempTest {
```

```
private void test1(A a){  
  
}  
public static void main(String[] args) {  
    TempTest t = new TempTest();  
    A a = new A();  
    t.test1(a); //这里传递的参数a就是按引用传递  
}  
}  
class A{  
    public int age = 0;  
}
```

### 3.3: 按引用传递的重要特点

传递的是值的引用，也就是说传递前和传递后都指向同一个引用（也就是同一个内存空间）。

示例如下：

```
第1行 public class TempTest {  
第2行     private void test1(A a){  
第3行         a.age = 20;  
第4行         System.out.println("test1方法中的age="+a.age);  
第5行     }  
第6行     public static void main(String[] args) {  
第7行         TempTest t = new TempTest();  
第8行         A a = new A();  
第9行         a.age = 10;  
第10行        t.test1(a);  
第11行        System.out.println("main方法中的age="+a.age);  
第12行    }  
第13行 }  
第14行 class A{  
第15行     public int age = 0;  
第16行 }
```

运行结果如下：

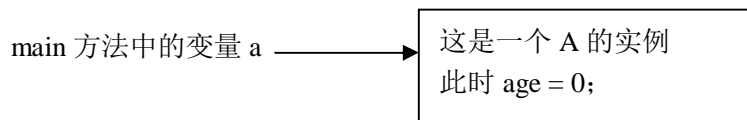
```
test1方法中的age=20  
main 方法中的 age=20
```

### 3.4: 理解按引用传递的过程——内存分配示意图

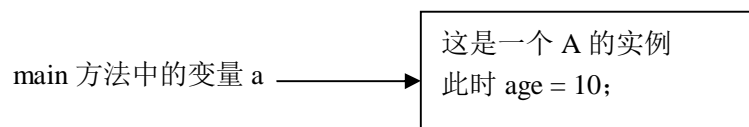
要想正确理解按引用传递的过程，就必须学会理解内存分配的过程，内存分配示意图可以辅助我们去理解这个过程。

用上面的例子来进行分析：

(1): 运行开始，运行第 8 行，创建了一个 A 的实例，内存分配示意如下：

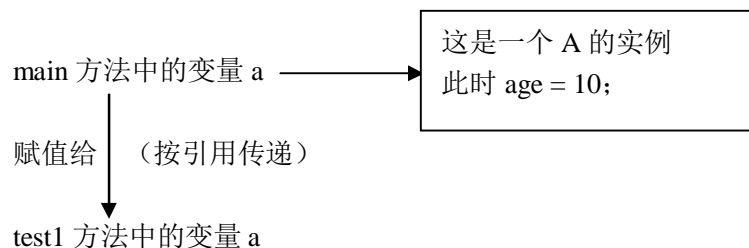


(2): 运行第 9 行，是修改 A 实例里面的 age 的值，运行后内存分配示意如下：

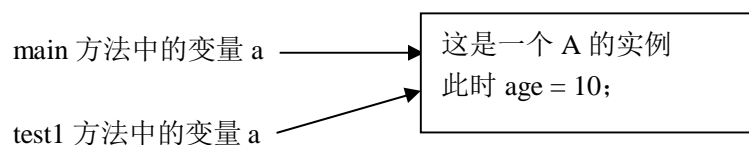


(3): 运行第 10 行，是把 main 方法中的变量 a 所引用的内存空间地址，按引用传递给 test1 方法中的 a 变量。请注意：这两个 a 变量是完全不同的，不要被名称相同所蒙蔽。

内存分配示意如下：

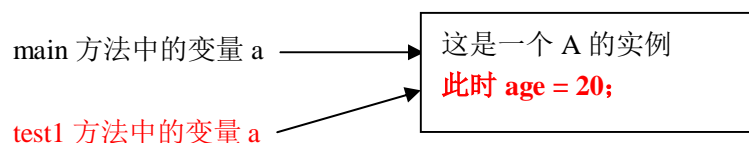


由于是按引用传递，也就是传递的是内存空间的地址，所以传递完成后形成的新的内存示意图如下：



也就是说：是两个变量都指向同一个空间。

(4): 运行第 3 行，为 test1 方法中的变量 a 指向的 A 实例的 age 进行赋值，完成后形成的新的内存示意图如下：



此时 A 实例的 age 值的变化是由 test1 方法引起的

(5): 运行第4行，根据此时的内存示意图，输出test1方法中的age=20

(6): 运行第 11 行，根据此时的内存示意图，输出 main 方法中的 age=20

### 3.5: 对上述例子的改变

理解了上面的例子，可能有人会问，那么能不能让按照引用传递的值，相互不影响呢？就是 test1 方法里面的修改不影响到 main 方法里面呢？

方法是在 test1 方法里面 new 一个实例就可以了。改变成下面的例子，其中第 3 行为

新加的:

```
第1行 public class TempTest {
第2行     private void test1(A a){
第3行         a = new A();//新加的一行
第4行         a.age = 20;
第5行         System.out.println("test1方法中的age="+a.age);
第6行     }
第7行     public static void main(String[] args) {
第8行         TempTest t = new TempTest();
第9行         A a = new A();
第10行        a.age = 10;
第11行        t.test1(a);
第12行        System.out.println("main方法中的age="+a.age);
第13行    }
第14行}
第15行class A{
第16行    public int age = 0;
第17行}
```

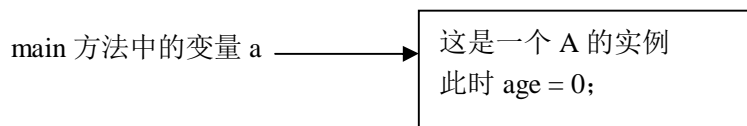
运行结果为:

```
test1方法中的age=20
main 方法中的 age=10
```

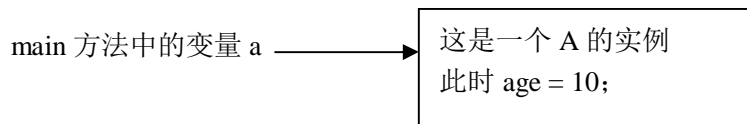
为什么这次的运行结果和前面的例子不一样呢, 还是使用内存示意图来理解一下

### 3.6: 再次理解按引用传递

(1): 运行开始, 运行第 9 行, 创建了一个 A 的实例, 内存分配示意如下:

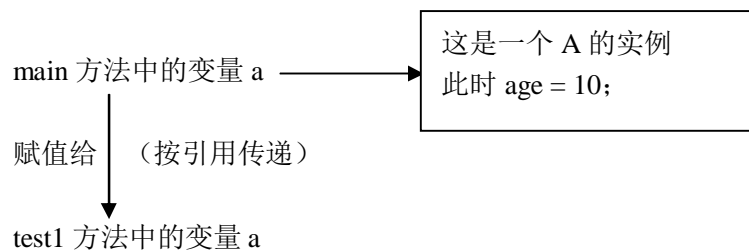


(2): 运行第 10 行, 是修改 A 实例里面的 age 的值, 运行后内存分配示意如下:

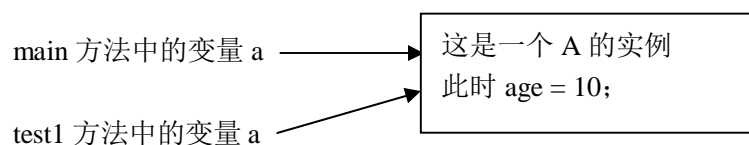


(3): 运行第 11 行, 是把 main 方法中的变量 a 所引用的内存空间地址, 按引用传递给 test1 方法中的 a 变量。请注意: 这两个 a 变量是完全不同的, 不要被名称相同所蒙蔽。

内存分配示意如下:

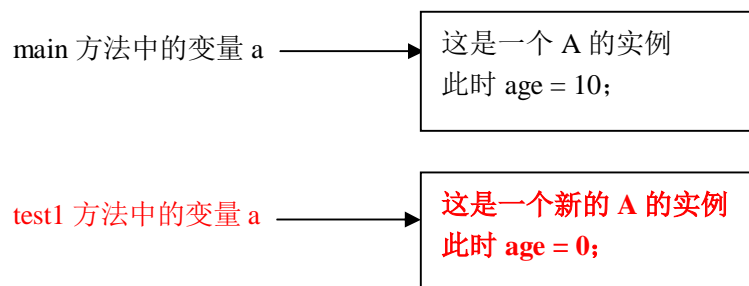


由于是按引用传递，也就是传递的是内存空间的地址，所以传递完成后形成的新的内存示意图如下：

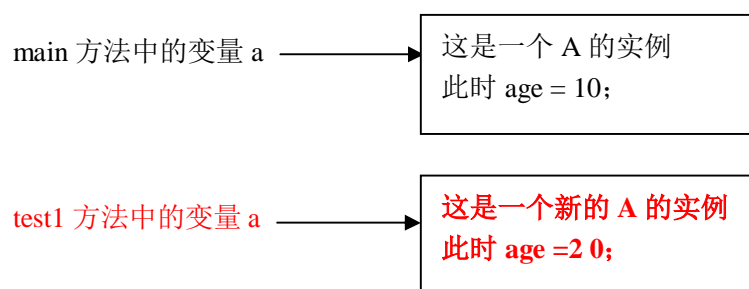


也就是说：是两个变量都指向同一个空间。

(4): 运行第 3 行，为 test1 方法中的变量 a 重新生成了新的 A 实例的，完成后形成的新的内存示意图如下：



(5): 运行第 4 行，为 test1 方法中的变量 a 指向的新的 A 实例的 age 进行赋值，完成后形成的新的内存示意图如下：



注意：这个时候 test1 方法中的变量 a 的 age 被改变，而 main 方法中的是没有改变的。

(6): 运行第5行，根据此时的内存示意图，输出test1方法中的age=20

(7): 运行第 12 行，根据此时的内存示意图，输出 main 方法中的 age=10

### 3.7: 说明

(1): “在 Java 里面参数传递都是按值传递”这句话的意思是: 按值传递是传递的值的拷贝, 按引用传递其实传递的是引用的地址值, 所以统称按值传递。

(2): 在 Java 里面只有基本类型和按照下面这种定义方式的 String 是按值传递, 其它的都是按引用传递。就是直接使用双引号定义字符串方式: `String str = “Java 私塾”;`