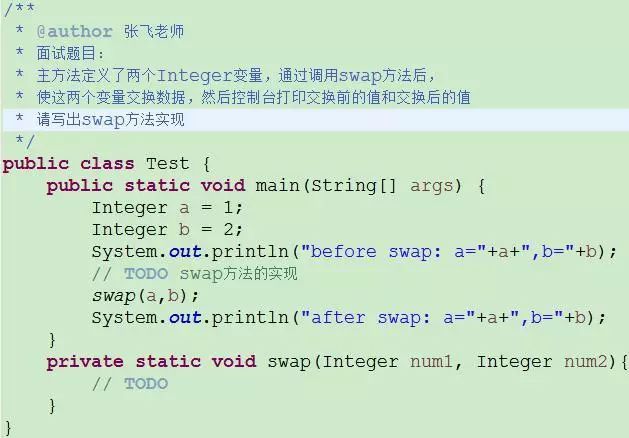
**Java大型互联网公司经典面试题，论JDK源码的重要性的无限思考**

*2017-11-28* *Java耕耘者* [Java后端技术](https://mp.weixin.qq.com/s/xf1USfUFBGYZZB68gbFHqg##)

论JDK源码的重要性：一道面试题引发的无限思考！大家在看到这个标题时想的是什么？小编我为什么要讲这个问题呢？

那我们就看一下这道面试题是什么呢？差不多是这样子的面试题



题目的意思是：定义了两个Integer类型变量，通过swap方法交换这两个变量的值。

看似简单的题目，是不是不知道从何下手，我猜想有些大家第一想到的是这样的解法：来看代码：

运行结果如下：

从结果来看是错误的，不能解决我们的问题。为什么？

**在分析之前，我们先介绍一下Java访问对象的方式。在 Java 堆中还必须包含能查找到此对象类型数据（如对象类型、父类、 实现的接口、方法等）的地址信息，这些类型数据则存储在方法区中。**

既然java栈中的是对象的引用，那么我们如何使用对象那，主流的访问方式有两种：**使用句柄和直接指针**。

**（1）使用句柄：**

如果使用句柄访问方式， Java 堆中将会划分出一块内存来作为句柄池，reference 中存储的就是对象的句柄地址，而句柄中包含了对象实例数据和类型数据各自的具体地址信息，如图：**（2）直接指针**

如果使用直接指针访问方式， Java 堆对象的布局中就必须考虑如何放置访问类型数据的相关信息， reference 中直接存储的就是对象地址，如图：这两种对象的访问方式各有优势，使用句柄访问方式的最大好处就是 reference 中存储的是稳定的句柄地址，在对象被移动（垃圾收集时移动对象是非常普遍的行为）时只会改变句柄中的实例数据指针，而 reference 本身不需要被修改。

使用直接指针访问方式的最大好处就是**速度更快**，它节省了一次指针定位的时间开销，由于对象的访问在 Java 中非常频繁，因此这类开销积少成多后也是一项非常可观的执行成本。

接着我们回到正题，这里也是今天要讲的第一个知识点：Java的传值在java中，有两种传值方式：**一种是按值传递，一种是引用传递！**

那么，按值传递意味着将当前的参数传递给方法的时候，方法中的变量接收的是传过来变量的副本值（相当于拷贝了一份值），因此，我们修改了方法里面的变量的值，并不会改变外面变量的值。

**引用传递：传递的是指向值的地址的指针**

那 么，请问大家，这里是按值传递还是引用传递？好，老司机告诉你们，这里是按值传递，为什么？Integer不是对象吗？ 对象传递不是传递的指针吗？大家有没有去看过Integer类的源码，看看这个类是怎么定义的，我们来看下，实际上面Integer使用的final定义 的，也就意味着通过Integer实例化的对象是不能改变的，跟String是不是差不多。所以这里的话，是传递的值，我们来画下图：

那么，我们首先看一下**Java运行时数据区域：**

我 们一般在开发中认为JVM不过有堆和栈两部分组成，但是实际的Java 虚拟机在执行 Java 程序的过程中会把它所管理的内存划分为若干个不同的数据区域。这些区域都有各自的用途，以及创建和销毁的时间，有的区域随着虚拟机进程的启动而存在，有些 区域则是依赖用户线程的启动和结束而建立和销毁。如下图：

Java中的内存主要分为两块把：堆和栈，栈存储变量本身，堆存储对象的值，然后通过栈执行堆内存地址来建立关系。

通过swap方法后：意味着，会同样创建两个变量num1和num2,他们的值是刚刚拷贝过来的ab的值，此时内存中时怎么变化的呢：

大家，知道为什么会有地址指针这个东西，主要是我们的堆内存他主要是存储的是一些对象，对象是最占内存的，为了能够节省对内从的空间，就出现了这种概念。好，讲到这里，至少大家应该清楚了一点：引用传递和按值传递的不同。

我们再来看，这个Integer他内部是如何赋值的，我们来看下：进入Integer类Ctrl+o搜索Integer构造方法：

然后我们发现这个value定义的是final类型的：

如果他有一个setValue()的方法的话，那么我们是是不是可以通过这个方法来改变值，但是Integer并没有提供。也就是说这种方法是行不通的，好，那么我们今天讲到第二个知识点：反射有没有人在做这个题目的时候有没有想过用反射来实现？

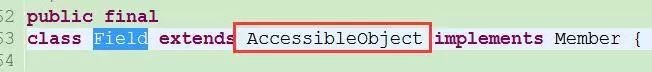
有 想过的，看有多少人有往这个方面去想，我们刚刚看到Integer类中存在一个value值变量吗？对吧，所以我们需要拿到这个value变量然后来改变 他的值，对吧，那么我们怎么来做，我们可以通过反射的方式拿到这个变量，这个Filed，然后去改变他的值，对吧。我们来看下怎么写：

理 论上来说，这种方式是一定能够实现我们的要求的.Run下：报错：“Class com.edu.example.test.Test can not access a member of class java.lang.Integer with modifiers "private final"”

报错了，是不是，那么这又是另外一个知识点：

**私有的成员属性是不能通过反射来赋值的！**

那么，如果要强攻，怎么办？实际上面，在java反射中，提供了一个叫设置访问权限的东西，我们进入Field类中看下：



然后他里面有一个setAccessible的方法：



这个方法就是用来设置成员属性访问权限的。我们看到最后是给obj.override=flag

那么我们在回过头来看下，Field的set方法：

这几行代码意味着，也就是说，如果override是false,就会调用Reflection.quickCheckMemberAccess(clazz, modifiers)来检查成员属性的访问权限。

所以说，我们再来看，这个时候是不是就可以通过设置setAccessible(true)为true来标志不需要访问权限的检查。这样就可以修改value的值了。对不。我们来试验下：

好，大家觉得这样没问题，结果如下：

结果是，a的值确实变了，但是b的值却没有变，首先说明通过这种方式确实可以改变值，但是为什么b的值没有变化呢？。请问为什么？我们再回过头来看看外面的方法，检查一下，我们定义了：

有没有发现什么问题？

Integer是不是一个封装类型，而他的值1,2，是不是一个int类型，是一个基本数据类型，那么这里是怎么赋值的呢？ 那么我们按照正常来写是不是这样子的：

int a = 1;

但是为什么使用Integer也不会报错了，好，这就讲到了我们又一个知识点：（笔记）

**Java中的装箱和拆箱**

**装箱**：把基本类型用它们相应的引用类型包装起来，使其具有对象的性质。int包装成Integer、float包装成Float;

**拆箱**：和装箱相反，将引用类型的对象简化成值类型的数据;

Integer a = 100; // 这是自动装箱  (编译器调用的是static Integer valueOf(int i))  
int  b = new Integer(100);  //这是自动拆箱

那么我们来实际看下，我们耳听为虚，眼见为实，我们来看下编译的字节码文件：

命令：javap -c Test.class

可以看到：

Jvm他自动做了装箱操作，看的清清楚楚对吧，对吧

好那么，我们来看下Integer.valueOf(1)：源码

意味着值大于IntegerCace.low小于IntegerCache.high的话：

会从IntegerCache中获取，也就是从缓存中取值。

那么我们来看下IntegerCache:

也就是说从-128到127直接的所有值，都是从缓存中获取。而缓存中的值，是什么时候放进去的，是jvm启动的时候就放进去了，然后分配好内存地址。

你 们有没有发现，就短短几行代码，怎么就有这么多知识，是不是都有点感觉不认识java了。很神奇吧，哈哈好，前面这两行代码我们分析完了对吧，好，然后， 然后我们把ab的值传进来，我们再来分析swap中的这段代码，好吧，精华部分就是这段代码了啊，这是精华部分，哈哈，我们来看：断点到这句

然后按F5进去看下，把IntegerCache里面的值全面拿出来放到notepat++

第一步：是不是需要获取num2的值，那么他从下标[2+128=130]IntegerCache中获取值为：130下标，也就是第131个数字为：2

第 二步：field.set(num1,num2),， 意味着第一步先获取num1在IntegerCache中的值IntegerCache[1+128] =1 ,然后会修改IntegerCache[num1]的值为num2从Integercache中获取到的值2， 也就是修改为：integerCache[129] = 2

第三步：下一行代码执行

此 时，再次拿出IntegerCache， 那么下标为129,130的值都变成了2， 此时tmp的值为1，那么从IntegerCache获取到的值为IntegerCache[1+128=129] ,也就是获取130行的数，也就是2，所以结果就是这样。实际上面和下面这个是一样的：

从这一句debug进去：发现走的缓存，然后从cache中第129个下标找到了。

所以，当我们的值是在【-127-128】的时候，他是从IntegerCache中获取的。其实，我们可以这样来验证一下：

结果为：true

结果为：false

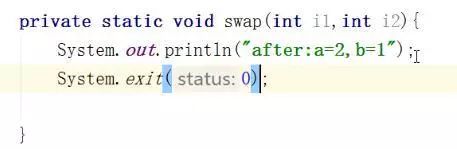
那么，这个当时我其实又遇到这个坑，被坑惨了是吧。哈哈哈。

那么我们怎么解决最后的问题：（最初的面试问题）

1.

2.

3.取巧的方式：



好啦。到此结束啦！