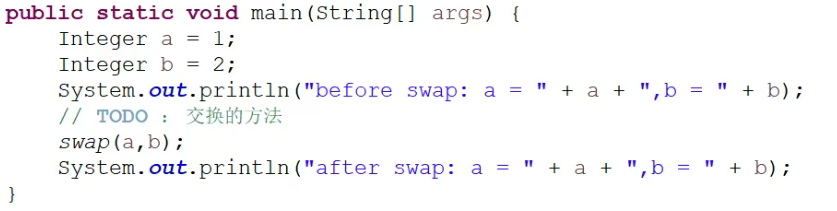
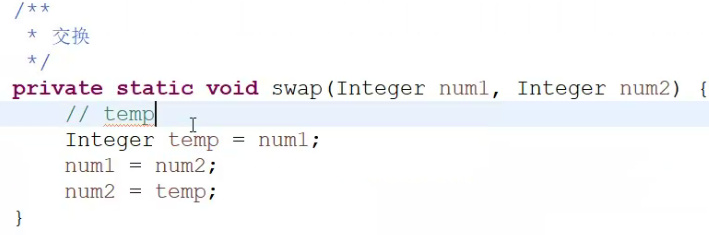
JDK源码探究-图灵学院

# 关于Integer源码的面试题目

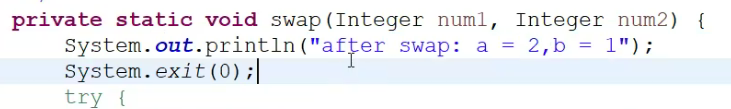
面试题目：保持不改变main方法，书写swap方法，实现两次输入的a和b数值交换。



错解1：值传递，并没有改变a和b的值。

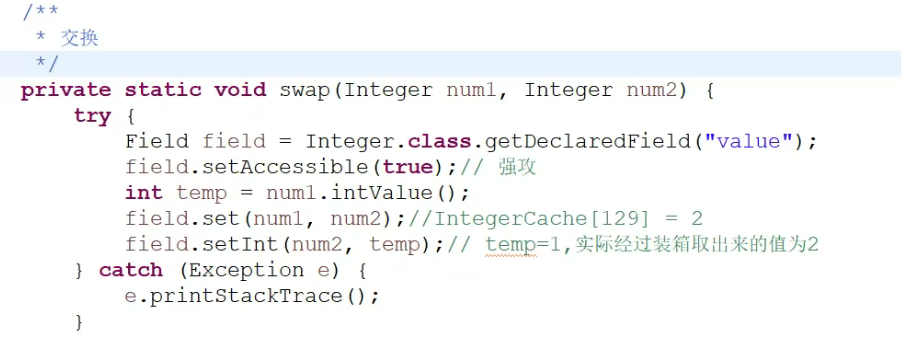


投机取巧的方法：**手动输出然后系统退出**。



正确的方法:

利用**反射机制和Integer的内部源码**。

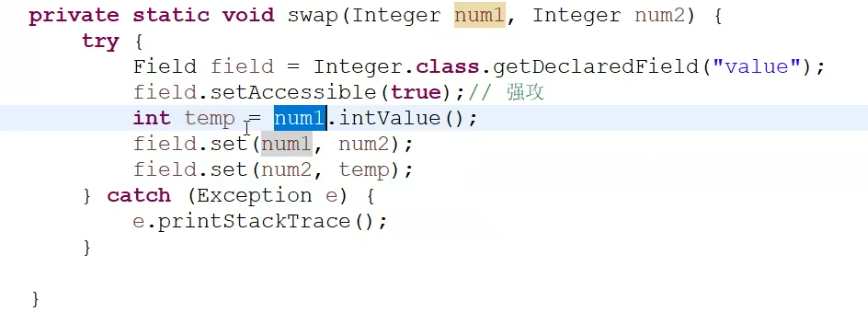


输出为：

a = 1; b = 2

a = 2; b = 1

注意:错解：

  
输出结果为：

a = 1; b = 2

a = 2; b = 2

剖析：

public static void main(String[] args) {

Integer a = 1;

Integer b = 2;

System.out.println("a = " +a+"; b = "+b);

swap(a,b);

System.out.println("a = " +a+"; b = "+b);

}

private static void swap(Integer a, Integer b) {

Field field = null;

try {

**field = Integer.class.getDeclaredField("value");**

**field.setAccessible(true);//强攻**

**int temp = a.intValue();**

**field.set(a, b);**

**field.setInt(b, temp);**

} catch (Exception e) {

e.printStackTrace();

}

}

关键是两个方法:Field的set(obj,obj) 和setInt(obj,int)。

对于Integer的value对应的Field，set设置的obj，如果是在cache的-128到127范围内，则改变的是cache的对象，如果不是该范围，改变的就是其他对象。如果两个都是set(obj,obj)方法，那么对于1和2值来说对应的cache索引为129和130都存放value为2的Integer对象了。

# Integer源码解读

## Integer的基本信息

public **final** class Integer **extends Number implements Comparable<Integer>**

**存在于java.lang包中，继承Number类，实现了Comparable接口，final修饰的。**

**注意：String、八种基本数据类型的封装类(Byte、Integer、Short、Long、Float、Double、Boolean、Character)都是final修饰的。**

## Integer有个属性value

可以通过反射机制获取该value属性对应的Field对象。

## Integer内部封装一个静态缓存类IntegerCache

源码：

/\*\*

**\* Cache to support the object identity semantics of autoboxing for values between**

**\* -128 and 127 (inclusive) as required by JLS.**

\*

\* The cache is initialized on first usage. **The size of the cache**

**\* may be controlled by the -XX:AutoBoxCacheMax=<size> option.**

\* During VM initialization, **java.lang.Integer.IntegerCache.high** property

\* may be set and saved in the private system properties in the

\* sun.misc.VM class.

\*/

private static class IntegerCache {

static **final** int low = -128;

static **final** int high;

static **final** Integer cache[];

static {

// high value may be configured by property

int h = 127;

String integerCacheHighPropValue =

sun.misc.VM.getSavedProperty("java.lang.Integer.IntegerCache.high");

**if (integerCacheHighPropValue != null) {**

**int i = parseInt(integerCacheHighPropValue);**

**i = Math.max(i, 127);**

**// Maximum array size is Integer.MAX\_VALUE**

**h = Math.min(i, Integer.MAX\_VALUE - (-low));**

**}**

high = h;

cache = new Integer[(high - low) + 1];

int j = low;

**for(int k = 0; k < cache.length; k++)**

**cache[k] = new Integer(j++);**

}

private IntegerCache() {}

}

解读：Integer内部封装的**IntegerCache内部类**，是用来对特定范围的数据自动装箱使用，对于Integer来说，缓存的最小的数据已经确定，low = -128；但是high可以通过**java.lang.Integer.IntegerCache.high**配置获取，如果没有配置，默认为127。**i = Math.max(i, 127);，可以看出high值最小必须为127。**

**IntegerCache**中有个Integer数组static final Integer **cache[]**; ，该数组的长度为**high-low+1**，通过静态代码块初始化该数组。

**for(int k = 0; k < cache.length; k++)**

**cache[k] = new Integer(j++);**

**注意：**cache这个Integer数组的索引与对应的value值相差128，即索引0处存放-128，索引100处存放-28，索引255处存放127。

## Integer的自动装箱原理

关键方法：valueOf(int i)，很多其他方法间接调用该方法。

/\*\*

\* Returns an {@code Integer} instance representing the specified

\* {@code int} value. If a new {@code Integer} instance is not

\* required, this method should generally be used in preference to

\* the constructor {@link #Integer(int)}, as this method is likely

\* to yield significantly better space and time performance by

\* caching frequently requested values.

\*

\* This method will always cache values in the range -128 to 127,

\* inclusive, and may cache other values outside of this range.

\*

\* @param i an {@code int} value.

\* @return an {@code Integer} instance representing {@code i}.

\* @since 1.5

\*/

**public static Integer valueOf(int i) {**

**assert IntegerCache.high >= 127;**

**if (i >= IntegerCache.low && i <= IntegerCache.high)**

**return IntegerCache.cache[i + (-IntegerCache.low)];**

**return new Integer(i);**

**}**

分成两部分，如果数值i在low和high之间，则直接返回IntegerCache的cache数据中的Integer对象，否则，new一个新的Integer对象。

**valueOf(String s,int radix)方法内部**就是调用valueOf(int i)方法实现的。

public static Integer valueOf(String s, int radix) throws NumberFormatException {

return Integer.valueOf(parseInt(s,radix));

}

## Integer对象的自动装箱原理

**Integer的自动装箱就是通过调用valueOf(int i)方法实现的。**

如果在low和high之间就从缓存数组中返回Integer对象；超过该范围就是new一个新的Integer对象。== 对于对象来说判断的是地址，对于int来说判断的是值。（若都为int，则判断值相等；若为Integer对象，判断对象地址）

笔试或者面试题目：请写出输出结果true or false？

public static void main(String[] args) {

**autoboxing(1);//true**

**autoboxing(-128);//true**

**autoboxing(127);//true**

**autoboxing(128);//false 有些设置了大于127的high可能是true**

**autoboxing(-129);//false**

}

private static void autoboxing(int i) {

Integer a = i;

Integer b = i;

System.out.println(a == b );//true or false?

}

因为Integer的缓存数组的low固定为-128，high一般默认为127，当然也可以设置比127更大的数，但是小于-128的数一定为false，在-128和127之间的一定为true， 大于127的不确定，默认情况下为false。

# 拓展Long、Short等装箱类

## Long、Short、Byte、Character都与Integer一样，具有缓存数组。

这种缓存行为不仅适用于Integer对象。我们**针对所有整数类型的类都有类似的缓存机制**。

有 ByteCache 用于缓存 Byte 对象；有 ShortCache 用于缓存 Short 对象；

有 LongCache 用于缓存 Long 对象；有 CharacterCache 用于缓存 Character 对象；

**Byte，Short，Long 有固定范围: -128 到 127**。**对于 Character, 范围是 0 到 127**。除了 Integer 可以通过参数改变范围外，其它的都不行。

只不过只有Integer可以通过配置设置high值，对于Long、Short、Byte来说，low和high都是固定的，都是-128到127。对于Character来说是0到127。

## Long

private static class LongCache {

private LongCache(){}

**static final Long cache[] = new Long[-(-128) + 127 + 1];**

static {

for(int i = 0; i < cache.length; i++)

cache[i] = new Long(i - 128);

}

}

public static Long valueOf(long l) {

final int offset = 128;

**if (l >= -128 && l <= 127) { // will cache**

**return LongCache.cache[(int)l + offset];**

**}**

**return new Long(l);**

}

## Short

private static class ShortCache {

private ShortCache(){}

static final Short cache[] = new Short[-(-128) + 127 + 1];

static {

for(int i = 0; i < cache.length; i++)

cache[i] = new Short((short)(i - 128));

}

}

public static Short valueOf(short s) {

final int offset = 128;

int sAsInt = s;

**if (sAsInt >= -128 && sAsInt <= 127) { // must cache**

**return ShortCache.cache[sAsInt + offset];**

**}**

**return new Short(s);**

}

## Byte

private static class ByteCache {

private ByteCache(){}

static final Byte cache[] = new Byte[-(-128) + 127 + 1];

**static {**

**for(int i = 0; i < cache.length; i++)**

**cache[i] = new Byte((byte)(i - 128));**

**}**

}

public static Byte valueOf(byte b) {

final int offset = 128;

return ByteCache.cache[(int)b + offset];

}

## Character

private static class CharacterCache {

private CharacterCache(){}

static final Character cache[] = new Character[127 + 1];

static {

for (int i = 0; i < cache.length; i++)

cache[i] = new Character((char)i);

}

}

public static Character valueOf(char c) {

**if (c <= 127) { // must cache**

**return CharacterCache.cache[(int)c];**

**}**

**return new Character(c);**

**}**