谈谈装箱与拆箱

一、何为包装类型

Java是一种面向对象的语言，但是它不是纯面向对象的。Java中存在基本数据类型，谈不上对象。为了向纯面向对象靠拢，Java5的时候推出了基本数据类型的包装类型。

基本数据类型与包装类型的对应关系如下：

二、何为装箱与拆箱

装箱就是将基本数据类型转化为包装类型，那么拆箱就是将包装类型转化为基本数据类型。

以基本数据类型int为例：

package day1119;

public class TestBox {

public static void main(String[] args) {

//自动装箱，底层其实执行了Integer a=Integer.valueOf(1);

Integer a = 1;

//自动拆箱，底层其实执行了int b=a.intValue();

int b = a;

}

}

Integer的valueOf（int i）方法可以将一个基本数据类型转化为对应的包装类型，即装箱方法。

而Integer的intValue（）方法则可以将一个包装类型转化为对应的基本数据类型，即拆箱方法。

三、装箱与自动装箱的区别

装箱：利用Integer的构造方法Integer（int value），即Integer c=new Integer(1);

自动装箱：或者叫隐式装箱，直接给Integer赋值，即Integer d=1,在编译的时候，会调用Integer.valueOf（）方法完成装箱。

相比而言，自动装箱可能比装箱具有更高的效率，体现在自动装箱的缓存上，下面从几道题目来讲自动装箱的缓存。

四、相关面试题目

第一题：以下代码的输出结果为？（==号两边如果都是引用类型的话，则判断它们是否指向同一个对象。如果都是基本数据类型的话，则判断它们的数值是否相等）

package day1119;

public class TestBox2 {

public static void main(String[] args) {

Integer a = 100;

Integer b = 100;

Integer c = 200;

Integer d = 200;

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

System.out.println(c == d);

}

}

也许有些人认为他们是四个各不相同的对象，两个式子都返回false。

实际运行后发现输出：

为什么一个是true，一个是false呢？

刚才我们知道，Integer a=100这条语句会触发自动装箱，而自动装箱的方法为Integer.valueOf（）方法，让我们去寻找这个方法，一探究竟。

观察Integer类的源码中的valueOf（）

/\*\*

\* 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) {

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

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

return new Integer(i);

}

可以看得出，当i的值位于[-128,127]的时候，会直接返回Integer缓存数组中相应对象的引用，如果i大于127或小于-128，会重新创建一个Integer实例，并返回。

那么第一条式子a和b的值都在缓存范围内，因此他们指向同一个对象，因此返回true。c和d的值不在范围内，都是通过new创建出来的，因此不是同一个对象，返回false。

注意：Byte、Short、Integer、Long、Character的valueOf（）实现机制类似。

其中相同值的Byte比较永远返回true，因为byte取值范围就是[-128,127]。

Short、Integer、Long的valueOf（）基本一样，i的范围都需要在[-128,127]。

Character中i的范围只要小于等于127即可，因为char最小值为0，本来就大于等于-128。

但是Float、Double中的valueOf（），永远返回新创建的对象，因为一个范围内的整数是有限的，但是小数却是无限的，无法保存在缓存中。

Boolean中只有两个对象，要么是true的Boolean，要么是false的Boolean，只要boolean的值相同，Boolean就相等。

第二题：以下代码的输出结果为？

package day1119;

public class TestBox3 {

public static void main(String[] args) {

Integer a = 1;

Integer b = 2;

Integer c = 3;

Long d = 2L;

Long e = 3L;

int f = 2;

//一旦有包装类型和数值类型判断==时，则触发包装类型的自动拆箱，转为数值类型的比较

System.out.println(new Integer(300) == 300);//返回true

//一旦有包装类型和数值类型发生运算时，则触发包装类型的自动拆箱，转为数值类型的运算

System.out.println(c == (a + f));//返回true

//一旦有包装类型和包装类型发生运算时，则触发包装类型的自动拆箱，转为数值类型的运算

System.out.println(c == (a + b));//返回true

//只有对象类型才有equals方法，因此首先a,b触发包装类型的自动拆箱，转为数值类型的运算。

//运算完，再将结果3自动装箱，Integer重写了equals,因此可以转为包装类型与包装类型的比较。

//当两边的包装类型不一致时，必定返回false。

//当两边的包装类型一致时,再进行拆箱，判断两者代表的数值是否相等。

System.out.println(c.equals(a + b));//返回true

//不同数据类型的数值进行运算，首先会将低精度的数据类型转化为高精度的数据类型，即自动类型转换。

//比如现在的int+long,会提升到long+long,再进行运算。

System.out.println(e == (a + d));//返回true

//==号两边类型不一致时，直接执行自动拆箱，比较之后的数值

System.out.println(e == (a + b));//返回true

//依次经历自动拆箱，自动类型转换、运算、自动装箱，类型比较，拆箱，数值比较

System.out.println(e.equals(a + d));//返回true

//依次经历自动拆箱，自动类型转换、运算、自动装箱，类型比较，两边类型不一致，直接返回false

System.out.println(c.equals(a + d));//返回false

}

}

五、总结

如果想要深入了解自动装箱拆箱的过程，必须得反编译class文件，了解底层编译的细节，才可以解除自己此方面的疑问。