**[从字节码角度分析Byte类型变量b++和++b](http://www.cnblogs.com/nailperry/p/4780354.html)**

<https://www.cnblogs.com/nailperry/p/4780354.html>

[**从字节码角度分析Byte类型变量b++和++b**](http://www.cnblogs.com/nailperry/p/4780354.html)

1. 下面是一到Java笔试题：

1 public class Test2

2 {

3 public void add(Byte b)

4 {

5 b = b++;

6 }

7 public void test()

8 {

9 Byte a = 127;

10 Byte b = 127;

11 add(++a);

12 System.out.print(a + " ");

13 add(b);

14 System.out.print(b + "");

15 }

16 }

2. 为方便分析起见，将打印的语句去掉，如下：

1 public void add(Byte b)

2 {

3 b = b++;

4 }

5 public void test()

6 {

7 Byte a = 127;

8 Byte b = 127;

9 add(++a);

10 add(b);

11 }

3. 将上述代码反编译，得到如下字节码：

1 public void add(java.lang.Byte);

2 Code:

3 0: aload\_1

4 1: astore\_2

5 2: aload\_1

6 3: invokevirtual #2 // Method java/lang/Byte.byteValue:(

7 )B

8 6: iconst\_1

9 7: iadd

10 8: i2b

11 9: invokestatic #3 // Method java/lang/Byte.valueOf:(B)

12 Ljava/lang/Byte;

13 12: dup

14 13: astore\_1

15 14: astore\_3

16 15: aload\_2

17 16: astore\_1

18 17: return

19

20 public void test();

21 Code:

22 0: bipush 127

23 2: invokestatic #3 // Method java/lang/Byte.valueOf:(B)

24 Ljava/lang/Byte;

25 5: astore\_1

26 6: bipush 127

27 8: invokestatic #3 // Method java/lang/Byte.valueOf:(B)

28 Ljava/lang/Byte;

29 11: astore\_2

30 12: aload\_0

31 13: aload\_1

32 14: invokevirtual #2 // Method java/lang/Byte.byteValue:(

33 )B

34 17: iconst\_1

35 18: iadd

36 19: i2b

37 20: invokestatic #3 // Method java/lang/Byte.valueOf:(B)

38 Ljava/lang/Byte;

39 23: dup

40 24: astore\_1

41 25: invokevirtual #4 // Method add:(Ljava/lang/Byte;)V

42 28: aload\_0

43 29: aload\_2

44 30: invokevirtual #4 // Method add:(Ljava/lang/Byte;)V

45 33: return

46 }

 4. 字节码很长，看着发怵，不用怕，我们将字节码分成两部分：add方法和test方法。

 5. 我们先来看add方法：

1 add方法局部变量表

2 下标： 0 1 2 3

3 标记： this 形参Byte b Byte型临时变量tmp Byte型临时变量tmp2

4 值 ： -128 -128 -127

5 public void add(java.lang.Byte);

6 Code:

7 0: aload\_1 // 局部变量表中下标为1的引用型局部变量b进栈

8 1: astore\_2 // 将栈顶数值赋值给局部变量表中下标为2的引用型局部变量tmp，栈顶数值出栈。

9 2: aload\_1 // 局部变量表中下标为1的引用型局部变量b进栈

10 3: invokevirtual #2 // 自动拆箱，访问栈顶元素b，调用实例方法b.byteValue获取b所指Byte

11 // 对象的value值-128，并压栈

12 6: iconst\_1 // int型常量值1进栈

13 7: iadd // 依次弹出栈顶两int型数值1(0000 0001)、-128(1000 0000)

14 //（byte类型自动转型为int类型）相加，并将结果-127(1000 0001)进栈

15 8: i2b // 栈顶int值-127(1000 0001)出栈，强转成byte值-127(1000 0001)，并且结果进栈

16 9: invokestatic #3 // 自动装箱：访问栈顶元素，作为函数实参传入静态方法Byte.valueOf(byte)，

17 // 返回value值为-127的Byte对象的地址，并压栈

18 12: dup // 复制栈顶数值，并且复制值进栈

19 13: astore\_1 // 将栈顶数值赋值给局部变量表中下标为1的引用型局部变量b，栈顶数值出栈。此时b为-127

20 14: astore\_3 // 将栈顶数值赋值给局部变量表中下标为3的引用型局部变量tmp2，栈顶数值出栈。此时tmp2为-127

21 15: aload\_2 // 局部变量表中下标为2的引用型局部变量tmp进栈，即-128入栈

22 16: astore\_1 // 将栈顶数值赋值给局部变量表中下标为1的引用型局部变量b，栈顶数值出栈。此时b为-128

23 17: return

总结一下上述过程，核心步骤为b = b++;分为三步：参考：[http://blog.csdn.net/brooksychen/article/details/1624753](http://blog.csdn.net/brooksychen/article/details/1624753" \t "_blank)

①把变量b的值取出来，放在一个临时变量里（我们先记作tmp）；

②把变量b的值进行自加操作；

③把临时变量tmp的值作为自增运算前b的值使用，在本题中就是给变量b赋值。

到此可得出结论，add方法只是个摆设，没有任何作用，不修改实参的值。

 6. 搞懂了add方法，我们接下来分析test方法：

这里需要说明两点：

（1）由于Byte类缓存了[-128,127]之间的Byte对象，故当传入的实参byte相同时，通过Byte.valueOf(byte)返回的对象是同一个对象，详见Byte源码。

（2）如果是实例方法（非static），那么局部变量表的第0位索引的Slot默认是用于传递方法所属对象实例的引用，在方法中通过this访问。详见：[http://wangwengcn.iteye.com/blog/1622195](http://wangwengcn.iteye.com/blog/1622195" \t "_blank)

1 test方法局部变量表

2 下标： 0 1 2

3 标记： this 形参Byte a Byte型临时变量b

4 值 ： -128 127

5 public void test();

6 Code:

7 0: bipush 127 // 将一个byte型常量值推送至操作数栈栈顶

8 2: invokestatic #3 // 自动装箱：访问栈顶元素，作为函数实参传入静态方法Byte.valueOf(byte)，

9 // 返回value值为127的Byte对象的地址，并压栈

10 5: astore\_1 // 将栈顶数值赋值给局部变量表中下标为1的引用型局部变量a，栈顶数值出栈。此时a为127

11 6: bipush 127 // 将一个byte型常量值推送至操作数栈栈顶

12 8: invokestatic #3 // 自动装箱：访问栈顶元素，作为函数实参传入静态方法Byte.valueOf(byte)，

13 // 返回value值为127的Byte对象的地址，并压栈。这里需要说明一点，

14 // 由于Byte类缓存了[-128,127]之间的Byte对象，故当传入的实参byte相同时，

15 // 通过Byte.valueOf(byte)返回的对象是同一个对象，详见Byte源码。

16 11: astore\_2 // 将栈顶数值赋值给局部变量表中下标为2的引用型局部变量b，栈顶数值出栈。此时b为127

17 12: aload\_0 // 局部变量表中下标为0的引用型局部变量进栈，即this，加载this主要是为了下面通过this调用add方法。

18 13: aload\_1 // 局部变量表中下标为1的引用型局部变量a进栈

19 14: invokevirtual #2 // 自动拆箱，访问栈顶元素a，调用实例方法a.byteValue获取a所指Byte

20 // 对象的value值127，并压栈

21 17: iconst\_1 // int型常量值1进栈

22 18: iadd // 依次弹出栈顶两int型数值1(0000 0001)、127(0111 1111)

23 //（byte类型自动转型为int类型）相加，并将结果128(1000 0000)进栈

24 19: i2b // 栈顶int值128(1000 0000)出栈，强转成byte值-128(1000 0000)，并且结果进栈

25 20: invokestatic #3 // 自动装箱：访问栈顶元素，作为函数实参传入静态方法Byte.valueOf(byte)，

26 // 返回value值为-128的Byte对象的地址，并压栈

27 23: dup // 复制栈顶数值，并且复制值进栈

28 24: astore\_1 // 将栈顶数值赋值给局部变量表中下标为1的引用型局部变量a，栈顶数值出栈。此时a为-128

29 25: invokevirtual #4 // 调用实例方法add:(Byte)，传入的实参为栈顶元素，也即a的拷贝，前面已经分析过了，该调用不改变a的对象值

30 // 该实例方法的调用需要访问栈中的两个参数，一个是实参，也即a的拷贝，一个是在第12步入栈的this。

31 28: aload\_0 // 局部变量表中下标为0的引用型局部变量进栈，即this，加载this主要是为了下面通过this调用add方法。

32 29: aload\_2 // 局部变量表中下标为2的引用型局部变量b进栈

33 30: invokevirtual #4 // 调用实例方法add:(Byte)，传入的实参为栈顶元素，也即b，前面已经分析过了，该调用不改变b的对象值

34 // 该实例方法的调用需要访问栈中的两个参数，一个是实参，也即b，一个是在第28步入栈的this。

35 33: return // 函数执行到最后，b所指对象的值没有改变，仍为127。

36 }

7. 综合以上分析，原问题的输出为-128 127

8. 小结：

通过以上分析，我们发现该题综合考察了Byte自动拆/装箱、Byte对象缓存、Java编译器对i=i++的特殊处理等等，相当有难度呀。