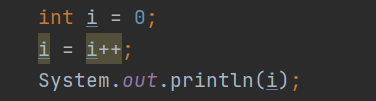
<https://blog.csdn.net/android_cai_niao/article/details/106027313>

i++相当于

int temp = i;

i = i + 1;

return temp;



Sout:i=0

示例代码

请先自己心算一下答案，然后找个本子记下来，然后再跟我后面的答案对比，看你能做对几道题，能做对两题以上的我喊你大哥！！

示例1

int i = 0;

i = i++;

System.out.println("i = " + i);

示例2

int a = 2;

int b = (3 \* a++) + a;

System.out.println(b);

示例3

int a = 2;

int b = a + (3 \* a++);

System.out.println(b);

示例4

int i = 1;

int j = 1;

int k = i++ + ++i + ++j + j++;

System.out.println(k);

示例5

int a = 0;

int b = 0;

a = a++;

b = a++;

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

示例答案

示例1：0

示例2：9

示例3：8

示例4：8

示例5：a = 1, b = 0

i++ 和 ++i原理

i++ 即后加加，原理是：先自增，然后返回自增之前的值

++i 即前加加，原理是：先自增，然后返回自增之后的值

重点：这是一般人所不知道的，记住：不论是前++还是后++，都有个共同点是先自增。

对于++i 就不说了，大多数人都懂，而对于 i++ 的原理，我用代码模拟其原理，如下：

int temp = i;

i = i + 1;

return temp;

这3句代码就是上面所说的那样：i++是先自增，然后才返回自增之前的值。

i++字节码分析

有很多的人写的文章上都是说i++是先返回i的值，然后再自增，这是错误，是先自增，然后再返回自增之前的值，你可能会问，这有区别吗？答案：有的。只要这个没理解对，则你在计算i++的相关问题时就有可能算错。

有的人可能又会问了，我凭什么相信你，你有什么证据证明i++是先自增，然后再返回自增之前的值吗？我还真去找过证据，我们把class的字节码搞出来，分析一下就知道了，证明如下：

public class Test {

void fun() {

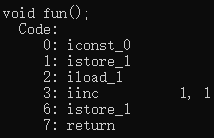
int i = 0;

i = i++;

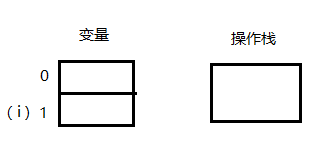
}

}

如上，我们写了一个超级简单的Test类。在cmd中输入这个命令（javap -c Test.class）以查看其生成的字节码，截图如下：



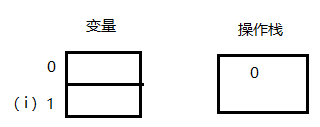
我们关注fun()方法这一段就可以了，如下：



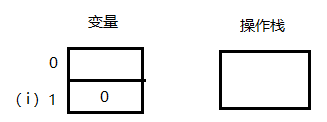
这就是fun()函数对应的字节码了，我们一行一行的来分析，首先我们要说两个概念，一个是变量，一个是操作栈，fun()方法中有两个变量，哎，不是只有一个变量i吗？怎么会有两个？要了解这个你需要去学习字节码的相关知识，这里我们不深究，我画图如下：

如上图，变量有两个，在位置0的变量是什么我们不要管，系统自动分配的，你要知道的是位置1的变量其实就是我们定义的变量i就行了，接下来，我们来一行行分析fun()方法对应的字节码：

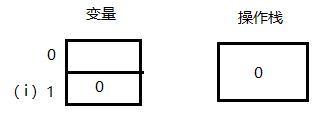
“ iconst\_0 ” i代表int类型，const代表常量，0就代表整数0，整句话的意思就是把int类型的常量0放入操作栈的栈顶中，图解如下：



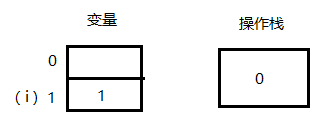
“ istore\_1 ” i代表int类型，store代表存储，1代表位置为1的变量，整句话的意思就是把操作栈中栈顶的值拿走，保存到位置为1的变量上，图解如下：



“ iload\_1 ” i代表int类型，load代表加载变量的值，1代表位置为1的变量，整句话的意思就是把位置为1的变量的值加载到操作栈的栈顶中，图解如下：

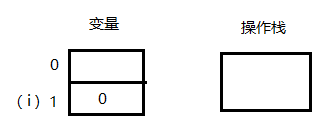


“ iinc 1, 1 ” i代表int类型，inc（increment）代表增加，这里还有两个1，前面的1代表对位置为1的变量，第2个1代表增加1，因为有i += 3这种自增操作，这种情况的话第2个数字会是3，即自增3（iinc 1, 3）。“iinc 1, 1” 整句话的意思就是把位置为1的变量的值增加1，图解如下：



注：自增操作不会改变操作栈中的值，所以变量i的值自增后变成了1，而操作栈中的值还是0。

“ istore\_1 ” i代表int类型，store代表存储，1代表位置1的变量，整句话的意思就是：把栈顶中的值拿走，保存到位置为1的变量中，图解如下：



所以，这几行字节码合起来看，i++不就是先自增，然后才返回自增之前的值嘛！！所以大家千万别搞错顺序了。 用代码理解的话，就相当于下面的代码：

int temp = i;

i = i + 1;

return temp;

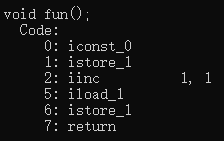
最后再把++i的字节码图也贴一下，大家可以根据我上面讲解的知识分析一下，就会知道++i和i++的区别了：

void fun() {

int i = 0;

i = ++i;

}



表达式原则

表达式有一个原则：一个变量也是表达式，多个表达式的加减法运算都是从左到右进行的

来看一下 if 语句的其中一种结构定义：

if (条件表达式) 语句;

1

用这个结构写个代码，如下：

boolean b = true;

int i = 0;

if(b) i++;

按照上面 if 语句的结构定义，if括号中是一个表达式，但是上面代码写了一个变量b进去，这是一个变量啊，怎么也能当成一个表达式么？没错，一个变量也是表达式。

记住这个重点：一个变量也是表达式，多个表达式的加减法运算都是从左到右进行的

讲到这里，估计有人会对这个运算顺序和乘法这些搞混了，示例如下：

int a = 0;

int b = a + a \* 2;

如上代码，按着我的说法，一个变量也是一个表达式，“b = a + a \* 2”这里a出现了两次，就是有两个a表达式，从左到右算的话先算a + a，这肯定不对啊，这不是我的意思哈，乘除法的优先级还是不能乱的，那应该先算a \* 2吗？也不对，应该是这样的：因为有乘法，所以a \* 2优先组成表达式，而不是a + a组成表达式，也就是说总体上可以分为两个表达式：“a” 表达式 和 “a \* 2” 表达式，这两个表达式相加肯定从左到右计算嘛，先算完a表达式的结果，再算a \* 2表达式的结果。你可能会想先算a和先算a \* 2有区别吗？答案是：有的，看完下面 的“示例3详解” 你就清楚了。

示例答案详解

示例1详解

int i = 0;

i = i++;

System.out.println("i = " + i); // 结果：0

先看i++，根据原理“先自增，然后返回自增之前的值”，i 自增后，i = 1，但是接着返回自增之前的值0，此时表达式变成 i = 0，0没有赋值给 i 时 i 的值是1，但是当把0赋值给 i 时，i 的值就又变成0了。因此 i = i++ 这句代码是做无用功，因为 i 的值最终还是和原来一样。

示例2详解

int a = 2;

int b = (3 \* a++) + a;

System.out.println(b); // 结果：9

int b = (3 \* a++) + a；a++后，a = 3，并返回自增之前的值2，所以此时表达式为：

int b = (3 \* 2) + a；此时a的值已经是3了，表达式又变为：

int b = (3 \* 2) + 3； 所以b = 9

示例3详解

int a = 2;

int b = a + (3 \* a++);

System.out.println(b); // 结果：8

这题和示例2几乎一样啊，只是换了一下顺序而已，为什么结果就不一样了呢?这就需要用到“表达式原则 了”：一个变量也是表达式，多个表达式的加减法运算都是从左到右进行的

int b = a + (3 \* a++)；按一般人的想法是先算 3 \* a++，a 先自增 a=3，然后返回自增之前的值2，所以此时表达式变为：

int b = a + (3 \* 2)； 此时a的值为3了，表达式又变为：

int b = 3 + (3 \* 2)；结果 b = 9

我们说一个变量也是表达式，多个表达式的加减法运算都是从左到右进行的，这个理论你可能不能深刻体会，但是如果我把代码稍微改一下你就能理解了，如下：

int b = (a \* 1) + (3 \* a++) 这个代码和 int b = a + (3 \* a++) 是一样的，没有区别，但是看(a \*1)你就很容易的知道要先算a \* 1这个表达式，表达式的结果为2。

所以，虽然 int b = a + (3 \* a++) 中前面的a只是一个变量，但他也是一个表达式，a这个表达式和(3 \* a++)这个表达式进行相加，多个表达式的运算都是从左到右进行的，所以先算a这个表达式，a表达式计算结果为2，所以表达式变成：

int b = 2 + (3 \* a++) 然后是a自增并返回自增之前的值2，所以表达式又变为：

int b = 2 + (3 \* 2)；所以结果为8。此时a的值为3

示例4详解

int i = 1;

int j = 1;

int k = i++ + ++i + ++j + j++;

System.out.println(k); // 结果：8

有了前面3条示例的详解，相信这一条大家就能自己解答了，可以先自己解答一下，看结果是不是8，不是的话，再来看我下面的讲解：

表达式原则说多个表达式的加减法运算都是从左到右进行的，这里的表达式有：i++、++i、++j、j++，都是加法，那我们就从左到右计算这4个表达式就OK了，如下：

1、先算i++，i++之后i的值为2，并返回++之前的值1，所以整个表达式可以变为：

   1 + ++i + ++j + j++; // 此时的i值为2

2、再计算++i，++i之后i的值为3，并返回3，所以整个表达式可以变为：

   1 + 3 + ++j + j++; // 此时i的值为3

3、再计算++j，++j之后j的值为2，并返回2，所以整个表达式可以变为：

   1 + 3 + 2 + j++; // 此时j的值为2

4、再计算j++，j++之后 j的值为3，并返回2，所以整个表达式可以变为：

   1 + 3 + 2 +2; // 结果为8，此时j的值为3

示例5详解

int a = 0;

int b = 0;

a = a++;

b = a++;

System.out.println("a = " + a + ", b = " + b); // a = 1, b = 0

到了第5题，好像已经没有难度了，大家应该都能解出来了，但是为了文章的完整性，我还是分解一下，大家应该自己先算一次，算不对再来看我的分解：

a = a++; a++之后a的值为1，并返回0，所以a的值由1又变回了0

b = a++; a++之后a的值为1，并返回0，0赋值给b，所以b为0，而a还是1哦！！

总结

i++ 即后加加，原理是：先自增，然后返回自增之前的值

++i 即前加加，原理是：先自增，然后返回自增之后的值

一个变量也是表达式，多个表达式的加减法运算都是从左到右进行的