# 运算符

运算符是一些特殊的符号，主要用于数学函数、一些类型的赋值语句和逻辑比较方面。Java中提供了丰富的运算符，如赋值运算符、算数运算符、比较运算符等，下面将详细介绍一下这些运算符。

## 1 赋值运算符

赋值运算符即“=”，是一个二元运算符（即对两个操作数进行处理），其功能是将右方操作数所含的值赋值给左方的操作数。语法格式如下：

变量类型 变量名 = 所赋的值；

左方必须是一个变量名，而右边所赋的值可以是任何数值或表达式，包括变量（如a、number）、常量（如123、‘book’）或有效的表达式（如45\*12）。

赋值运算符“=”处理时会先取得右方表达式处理后的结果，因此一个表达式中若含有两个以上的“=”运算符，会从最右方的“=”开始处理。

注意:因为java中的各种数据说白了都是在内存中的一块区域,所以赋值运算符说白了只是传递了一个指针指向;

【例1.1】创建类EvaluationPractice，在类中进行一些赋值操作

class EvaluationPractice{

public static void main(String[] args){

int ax = 1 , bx = 9 ; //为1个int型变量名为ax的变量赋值,初始化值为1

int cx = ax = bx - ax ; //当一个表达式中若有两个以上的赋值运算符,那么会先从最右边算起,当然我们在程序代码中并不建议这么做.因为这不容易使得代码清晰理解

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

System.out.print("ax = "+ax);

}

}



赋值运算符除了“=”以外，还有其他几个（+=, -=, \*=, /=, %=），我们把它们放在算术运算符中介绍。

## 2 算术运算符

Java中的算数运算符主要有+（加号）、-（减号）、\*（乘号）、/（除号）和%（求余），它们都是二元运算符。Java中算术运算符的功能及使用方式如下所示。



其中“+”和“-”运算符还可以作为数据的正负符号，如+5、-7。

注意：除法运算时，要记住0不可以作为除数，否则会出现运行时异常ArithmeticException。如int a = 5/0；系统会报出上述异常。

如果对负数取模，可以把模数负号忽略不记，如：5%-2=1。但被模数是负数就另当别论。对于除号“/”，它的整数除和小数除是有区别的：整数之间做除法时，只保留整数部分而舍弃小数部分。 例如：int x=4150;x=x/1000\*1000; x的结果是？“+”除字符串相加功能外，还能把非字符串转换成字符串 ，例如：System.out.println("5+5="+5+5);//打印结果是？

【例2.21】创建ArithmeticPractice类，进行一些操作

|  |
| --- |
| **public** **class** ArithmeticPractice {  **public** **static** **void** main(String[] args){  **int** ax = 3 , bx = 2 , hx , jx = -1 , rx = -5;  **byte** cx = (**byte**)(ax + bx) ;  **long** dx = bx - ax ;  **float** ex = ax%bx ;  System.***out***.println("ax除以bx取余 = "+ex);  ex = ax%jx ;  //取余可以快速用来判断一个数的奇偶性,如a%2=0,那么a为偶数,反之为奇数  System.***out***.println("当被模数是负数时:"+ex);  ex = rx%bx ;  System.***out***.println("当模数是负数时:"+ex);//当模数是负数时,要考虑符号  ex = rx%-3 ;  System.***out***.println("当模数和被模数都为负数时会先将被模数的符号撇去然后运" +  "算，实际在取模运算中只考虑模数的取值符号，正则正，负则负："+ex);  **int** fx = ax \* bx ;  hx = 0 ;  //int gx = ax/hx ; 除数不能为0,否则会报错误,java.lang.ArithmeticException  **int** gx = ax / bx ; //除的结果为整数,余数会舍掉  System.***out***.println("ax+bx = "+cx);  System.***out***.println("bx-ax = "+dx);  System.***out***.println("ax\*bx = "+fx);  System.***out***.println("ax/bx = "+gx);  System.***out***.println("5+5 ="+5+5);//+除了可以做加法外,还可以把非字符串转换成字符串  System.***out***.println(4150 / 1000 \* 1000);  System.***out***.println("除法运算的符号则和正常的没什么不同" + rx/bx + "," + rx/jx);  }  } |
| ax除以bx取余 = 1.0  当被模数是负数时:0.0  当模数是负数时:-1.0  当模数和被模数都为负数时会先将被模数的符号撇去然后运算，实际在取模运算中只考虑模数的取值符号，正则正，负则负：-2.0  ax+bx = 5  bx-ax = -1  ax\*bx = 6  ax/bx = 1  5+5 =55  4000  除法运算的符号则和正常的没什么不同-2,5 |

赋值运算符中除了“=”还有“+=”,“-=”,“ \*=”,“ /=”,“ %=”，这些运算符会先完成赋值运算符前面的运算然后再赋值，这些运算符具有默认强制类型装换功能，我们做几个事例，帮助理解，int a,b,c; a=b=c =3; int a = 3; a+=5;等同运算a=a+5;

思考：short s = 5; s=s+2;(编译出错) s+=2;(正常) 有什么区别？(可以发现前者编译出错,这是由于参与运算时,s会转换成int类型,运算结果也是int类型,int类型赋值给short需要强制转换,所以编译出错;后者则是由于+=有默认的强制类型转换功能,它其实是 s = (short)(s + 2)的简写;)

【例2.22】我们在EvaluationPractice类中，作如下测试

//short s = 5 ;

//s = s + 2 ; //赋值运算符左边的变量s是short型,右边的在做运算时会先转换成默认int型,运算结果也是int型,而把右边的int型转换为short型,这是由高精度向低精度转换,会损失精度--编译不通过

//s+=2 ; //s+=2 在short类型下，它会强制转换成short类型---正常

//System.out.println(s); //运算结果为7

byte s = 5 ;

s+=123;

/\*8位byte类型二进制的范围为-128到127，因为首位为符号位，所以正数到127-01111111，就到头了，所以正数最大为127；-128的二进制位128的补码10000000，因为“正数128的二进制为10000000”，这是关键点，有人会想“正数128的二进制为10000000”这不负数吗，其实不然，只是byte类型首位为符号位而引起的，否则正数最大也不会是127，\*/

System.out.print("损失精度："+s);



## 3 自增和自减运算符

自增、自减运算符是单目运算符，可以放在操作员之前，也可以放在操作元之后。操作元必须是一个整型或浮点型变量。放在操作元前面的自增、自减运算符，会先将变量的值加1（减1），然后再使该变量参与表达式的运算；放在操作元后面的自增、自减运算符，会先使变量参与表达式的运算，然后再将该变量加1（减1）。实例代码如下：

++a(--a) //表示在使用变量a之前,先使a的值加(减)1

a++(a--) //表示在使用变量a之后,使a的值加(减)1

【例2.31】我们在Demo类中，进行下面一些验证

class Demo{

public static void main(String[] args){

int ax = 3 , bx = 2 , jx = -1;

ax = bx++ + jx ;

System.out.println("前加加和后加加的区别:"+ax+"......"+bx+"......"+ax);

bx = ++ax + jx ;

System.out.println("前加加和后加加的区别:"+ax+"......"+bx+"......"+ax);

}

}



在我们第一次运算时bx先进行和jx的运算,然后赋值给ax,最后进行了一次自加加,这时的输出结果如同第一次输出时一样,然后在做第二次运算时,ax这时先进行一次自加加,然后再进行和jx的运算,最后结果赋值给bx

再举一个例子如下,

|  |
| --- |
| bx = ax ++;  System.***out***.println(bx + “,” + ax);//2,3 |

出现这种情况我们可以进行如下几个步骤的分析,

* 将ax读入局部变量表1,此时为2
* 将ax的值赋值给bx,将bx的值存入局部变量表2
* 将ax的值自增一次,此时ax变为3

注意:

* 自增和自减不是原子的操作,实际上是由三个操作构成的,读入数据,运算,写入数据,所以它不是线程安全的,这点在线程中再详细讲解;
* 自增和自减运算会改变变量的值,所以它们的操作数不能是数值,如4++就不是一个合法的语句。
* 由于自增自减会使人困惑，而且稍不注意会带来不易察觉的bug，所以不鼓励使用。

## 4 比较运算符

比较运算符属于二元运算符,用于程序中的变量和变量之间、变量和常量之间以及其他类型的信息之间的比较。比较运算符的运算结果是boolean型，当运算符对应的关系成立时，运算结果是true，否则结果是false。比较运算符通常用在条件语句中来作为判断的依据。比较运算符的种类和用法如下表所示：



比较运算符里要注意==和=我们要注意区分开,一个是等于,一个是赋值是不一样的

【例2.41】创建ComparisonOperatorPractice类,进行一些操作

class ComparisonOperatorPractice{

public static void main(String[] args){

int s1 = 5 , s2 = 6 ;

System.out.println("5>6这是真的吗:"+(s1>s2));

System.out.println("5<6这是真的吗:"+(s1<s2));

System.out.println("5等于6是真的吗:"+(s1==s2));

System.out.println("5大于等于6是真的吗:"+(s1>=s2));

System.out.println("5小于等于6是真的吗:"+(s1<=s2));

System.out.println("5是不等于6的:"+(s1!=s2));

System.out.println("字符S和s是不相同的:"+('S'!='s'));

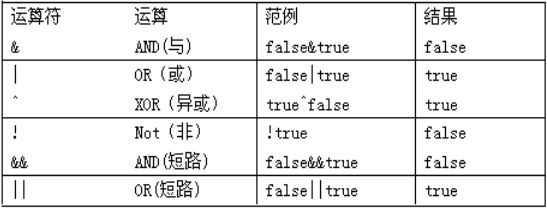
}

}



## 5 逻辑运算符

返回类型为布尔值的表达式，如比较运算符，可以被组合在一起构成一个更复杂的表达式。这是通过逻辑运算符来实现的。逻辑运算符包括&&(&)(逻辑与)、||（|）（逻辑或）、！（逻辑非）和^(逻辑异或)，**返回值为布尔类型的表达式，操作元也必须是boolean类型数据**。与比较运算符相比，逻辑运算符可以表达更加复杂的条件，如连接几个关系表达式进行判断（比如我们数学中有这么个表达式在Java中不可以写成3<x<6，应该写成x>3 & x<6）。在逻辑运算符中，除了“！”是一元运算符外，其余的都是二元运算符，用法如下：



在Java中，逻辑运算符“&&”与“&”都表示“逻辑与”，那么它们之间的区别在哪里呢？从上面的表中可以看出，当两个表达式都为true时，逻辑与的结果才会是true，使用逻辑运算符“&”会判断两个表达式；而逻辑运算符“&&”则是针对boolean类型进行判断,当第一个表达式为false时则不去判断第二个表达式,直接输出结果。使用“&&”可以节省计算机判断的次数。通常将这种在逻辑表达式中从左端的表达式中从左端的表达式可推断出整个表达式的值称为“短路”，而那些始终执行逻辑运算符两边的表达式称为“非短路”。“&&”属于“短路”运算符，而“&”则属于“非短路”运算符。“||”和“|”也是如此。

异或( ^ )与或( | )的不同之处是：当左右都为true时，结果为false。

【例2.51】创建LogicalOperatorPractice类，进行上述一些验证

class LogicalOperatorPractice{

public static void main(String[] args){

boolean b1 = 3 > 5 , b2 = 3 < 5 ;

System.out.println("逻辑单与,真真为真,真假为假,假真为假,假假为假:"+(b2&b2)+".."+(b2&b1)+".."+(b1&b2)+".."+(b1&b1));//单&时，左边无论真假，右边都进行运算；

System.out.println("逻辑双与,真真为真,真假为假,假真为假,假假为假:"+(b2&&b2)+".."+(b2&&b1)+".."+(b1&&b2)+".."+(b1&&b1));//双&时，如果左边为真，右边参与运算，如果左边为假，那么右边不参与运算。

System.out.println("逻辑单或,真真为真,真假为真,假真为真,假假为假:"+(b2|b2)+".."+(b2|b1)+".."+(b1|b2)+".."+(b1|b1));//单|时，左边无论真假，右边都进行运算；

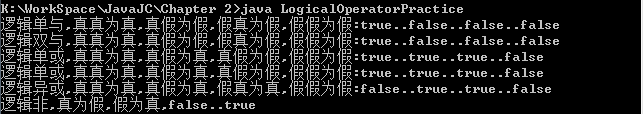
System.out.println("逻辑双或,真真为真,真假为真,真假为假,假假为假:"+(b2||b2)+".."+(b2||b1)+".."+(b1||b2)+".."+(b1||b1));//双||时，如果左边为真，右边不参与运算，如果左边为假，那么右边参与运算。

System.out.println("逻辑异或,真真为假,真假为真,假真为真,假假为假:"+(b2^b2)+".."+(b2^b1)+".."+(b1^b2)+".."+(b1^b1));

System.out.println("逻辑非,真为假,假为真,"+!b2+".."+!b1);

}

}



|  |
| --- |
| //注意短路运算符有时候可以避免出现一些异常,如下所示  **int** x = 0;  **int** y = 2;  System.***out***.println(x != 0 && 1 / x > x + y);//false |

在进行运算时,由于x=0，所以短路运算左边为false，所以后面不再进行运算，这就避免了1/0,避免了ArithmeticException。

## 6 三元运算符

三元运算符是Java中唯一一个三目运算符，其操作元有3个，第一个是条件表达式，其余的是两个值，条件表达式成立时运算取第一个值，不成立时取第二个值。实例代码如下：

Condition ？ expression1 ： expression2；

表达式：是指用运算符连接起来的，符合java语法的式子。

Java表达式可以分为如下几种：

算数表达式：20+10 关系表达式：800>200 逻辑表达式：10>5 && 100<80 赋值表达式：a=10

三元运算符用于判断，等价于if...else语句如下：

boolean a; //声明boolean变量

If(20 < 45) //将20 < 45作为判断条件

a = true; //条件成立将true赋值给a

else

a = false; //条件不成立将false赋值给a

当表达式”20 < 45”的运算结果返回真时，则boolean型变量a取值true;当表达式”20 < 45”返回假时,则boolean,则boolean型变量a取值false,此例的结果是true。

【例2.61】创建TernaryOperatorPractice类，并进行一些运算,获取两个数中的最大数

class TernaryOperatorPractice{

public static void main(String[] args){

int i = 23 , j = 45 ;

int z = i > j ? i : j ;

System.out.println(z);

}

}



## 7 位运算符

位运算符除按位与和按位或运算符外，其他只能用于处理整数的操作数。位运算是完全针对位方面的操作。整型数据在内存中是以二进制的形式表示，如int型变量7的二进制表示是00000000 00000000 00000000 00000111。

左边最高位是符号位，最高数是0表示正数，若为1则表示负数。负数采用补码表示，如-8的二进制表示为11111111 11111111 11111111 11111000.这样就可以对整数数据进行按位运算。Java语言提供的位运算符主要有下面几种。

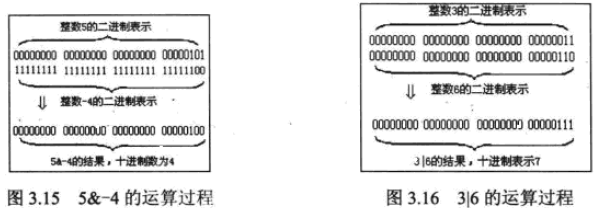


### 7.1 “按位与”运算

“按位与”运算的运算符为“&”，是双目运算符。其运算的法则是：如果两个操作数对应位都是1，则结果位才是1，否则为0.如果两个操作数的精度不同，则结果的精度与精度高的操作数相同，如图3.15所示。

### 7.2 “按位或”运算

“按位或”运算的运算符为“|”，是双目运算符。其运算法则是：如果两个操作数对应位都是0，则结果位才是0，否则为1.如果两个操作数的精度不同，则结果的精度与精度高的操作数相同，如图3.16所示。

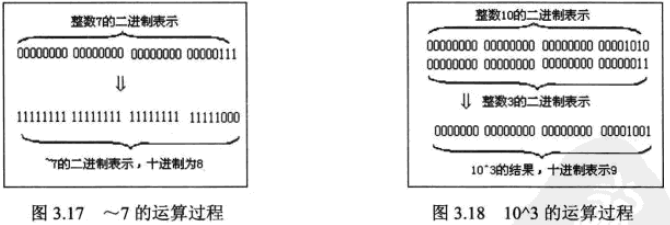


### 7.3 “按位取反”运算

“按位非”运算也称“按位取反”运算，运算符为“~”，是单目运算符。其运算法则是：将操作数二进制中的1全部修改为0,0全部修改为1，如图3.17所示。

### 7.4 “按位异或”运算

“按位异或”运算的运算符是“^”，是双目运算符。其运算法则是：当两个操作数的二进制表示相同（同时为0或同时为1）时，结果为0，否则为1。若两个操作数的精度不同，则结果数的精度与精度高的操作数相同，如图3.18所示。



### 7.5 移位操作

Java语言中的移位运算符有3种，其操作的数据类型只有byte、short、char、int和long5种。

* 左移运算符“<<”。所谓左移运算符，就是将左边的操作数在内存中的二进制数据左移右边操作数指定的位数，左边移空的部分补0.示例代码如下：

48 << 1; //将48的二进制数向左移1位

* 右移运算符”>>”，右移则复杂一些，当使用“>>”符号时,如果最高位是0,左移空的位就填入0;如果最高位是1，右移空的位就填入1，使用方法与左移类似。示例代码如下：

48 >> 1; //将48的二进制数向右移1位

0 1 1 0 0 0 0 0 96的二进制位

左移一位

0 0 1 1 0 0 0 0 48的二进制位

右移一位

0 0 0 1 1 0 0 0 24的二进制位

* 无符号右移运算符”>>>”。Java还提供了无符号右移运算符“>>>”，不管最高位是0还是1，左移空的高位都填入0。

移位能让用户实现整数除以或乘以2的n次方的效果，例如，y<<2与y\*4的结果相同;y>>1的结果与y/2的结果相同。总之，一个数左移n位,就是将这个数乘以2的n次方;一个数右移n位，就是将这个数除以2的n次方。

【例2.71】创建ShiftOperatorPractice类，进行移位运算符的验证操作

class ShiftOperatorsPractice{

public static void main(String[] args){

System.out.println("46&97的结果为:"+(46&97)); //46的二进制表示00000000 00000000 00000000 00101110

System.out.println("46|97的结果为:"+(46|97)); //97的二进制表示00000000 00000000 00000000 01100001

System.out.println("46按位取反的结果为:"+~46);

System.out.println("46^97的结果为:"+(46^97));

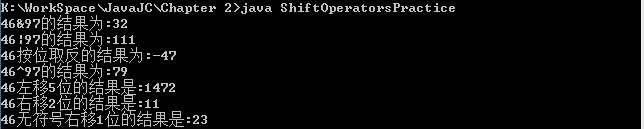
System.out.println("46左移5位的结果是:"+(46<<5));

System.out.println("46右移2位的结果是:"+(46>>2));

System.out.println("46无符号右移1位的结果是:"+(46>>>1));

}

}



思考题: 最有效率的方式算出2乘以8等于几？

对两个整数变量的值进行互换(不需要第三方变量)

【例2.72】创建类ShiftOperatorDemo，进行如下操作；

class ShiftOperatorDemo{

public static void main(String[] args){

int A = 23 , B = 46 ; //46的二进制:00000000 00000000 00000000 00101110

A = A ^ B ; //23的二进制:00000000 00000000 00000000 00010111

B = A ^ B ; //A^B 00000000 00000000 00000000 00111001

A = A ^ B ;

System.out.println("A="+A+","+"B="+B);

System.out.println("有效率的方式算出2乘以8的方法为2<<3:"+(2<<3));

}

}



思考题:移位运算符操作数是负数,那么该如何处理呢?下面我们通过一个例子来分析一下:

【例2.73】将2左移-3位，计算结果，并进行分析：

|  |
| --- |
| System.***out***.println(2<<-3); //1073741824 |

为什么会出现这样的结果呢?

在计算机系统中，数值一律用补码来表示和存储。原因在于，使用补码，可以将符号位和数值域统一处理；同时，加法和减法也可以统一处理。此外，补码与原码相互转换，其运算过程是相同的，不需要额外的硬件电路。

int 是32位，并且位运算底层用到补码，举个例子那么在位移<<或者>>右边的值i要用补码，并且因为int是32位的，所以右边位移i只取补码的后6位(6位中第一位作为符号位看待,移位数字为符号位后面5位数字,5位的最大值是32)，那么到了这里就可以对上面的代码进行解释了,上面-3的补码的后6位是111101,所以实际移位是16+8+4+1=29位,所以结果就是2\*2^29=1073741824了;

那么由此我们也可以发现2<<31的结果与2<<-1的结果相同,等等类似的,这是因为它们的补码后六位都相同,所以结果也都一样,由此我们也可以推断,实际上移位运算依赖的就是第二个操作数的补码的后六位（针对int类型）.

警告：移位运算符的右操作数要完成%32的运算（除非左操作作数是long类型，在这种情况下需要对右操作数%64）。例如1<<35的值等同于1<<3或8;

### 7.6  位移运算符练习

[例1]计算一个数的2的n次方;除以一个数的n次方

|  |
| --- |
| fun4(2, 4);       /\* 计算n次方 \*/  **public** **void** fun4(**int** i,**int** n){           System.*out*.println(i+"的"+n+"次方是:"+(i<<n));           System.*out*.println(32>>3);       } |



[例2]求一个数的绝对值

(a^(a>>31))-(a>>31)

先整理一下使用位运算取绝对值的思路：若a为正数，则不变，需要用异或0保持的特点；若a为负数，则其补码为源码翻转每一位后+1，先求其原码，补码求原码和原码求补码方法一致,符号位不变，数值位按位取反,末位再加1，此时需要使用异或1具有翻转的特点。

任何正数右移31后只剩符号位0，最终结果为0，任何负数右移31后也只剩符号位1，溢出的31位截断，空出的31位补符号位1，最终结果为-1.右移31操作可以取得任何整数的符号位。

那么综合上面的步骤，可得到公式。a>>31取得a的符号，若a为正数，a>>31等于0，a^0=a，不变；若a为负数,a>>31等于-1 ，a^-1翻转每一位.

小结:在日常的java开发中位运算使用的不是很常见，但是面试或考试中会有涉及的地方，虽然不是决定项，但却是加分项，说明对计算机语言有最起码的了解。而且在高级算法中，位运算往往能优化算法运行效率，减少运行时间。再比如，有一张全是选择题或是勾选题（类似判断）的试卷，你是使用每个选项一条记录的形式保存答案还是使用一个二进制对应的整数来保存答案？就像是英语考试中的答题卡：



每个题目有4个选项，每个选项有两个状态：选、不选（1、0），那么此时是不是可以使用4位二进制数来表示某题的答案呢？

|  |
| --- |
| fun5(-6);           fun5(7);  /\*\* 求一个数的绝对值\*/  **public** **void** fun5(**int** a){           System.*out*.println("绝对值:"+((a^(a>>31))-(a>>31)));       } |



[例3]求平均值,比如有两个int类型变量x、y,首先要求x+y的和，再除以2，但是有可能x+y的结果会超过int的最大表示范围，所以位运算就派上用场啦。(x&y)+((x^y)>>1);

|  |
| --- |
| fun6(999456789, 1234567898);  /\*求两个数的平均值\*/  **public** **void** fun6(**int** x,**int** y){           System.*out*.println("平均值:"+((x&y)+(x^y)>>1));       } |



[例4] 对于一个大于0的整数，判断它是不是2的几次方 ((x&(x-1))==0)&&(x!=0)；

|  |
| --- |
| /\*\* 判断一个数是不是2的n次方\*/  **public** **void** fun7(){  **int** x = 98768676;           System.*out*.println(((x&(x-1))==0)&&(x!=0));           x = 32;           System.*out*.println(((x&(x-1))==0)&&(x!=0));       } |



## 8 运算符优先级

Java中的表达式就是使用运算符连接起来的符合Java规则的式子,运算符的优先级决定了表达式中运算执行的先后顺序。各运算符的优先级由高到低如（增量和减量运算符==》算术运算符==》比较运算符==》逻辑运算符==》赋值运算符）所示；

各运算符之间大致的优先级由上列出，但如果两个符号属于同一运算符又怎么区分优先级呢？Java中各符号的优先级如下列表所示：



如果两个运算有相同的优先级，那么左边的表达式要比右边的表达式先被处理。在编写程序时尽量使用括号运算符来限定运算次序，以免产生错误的运算顺序。

[单目运算符](https://www.baidu.com/s?wd=%E5%8D%95%E7%9B%AE%E8%BF%90%E7%AE%97%E7%AC%A6&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3nHcvuW6LrHmdnAnvuHR40ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWTknjRsnWm3" \t "http://zhidao.baidu.com/_blank)是指：运算对象只有一个的运算符；如：取正（+）、取负（-）、取反（^）、或（|）、与（&）等等；  
[双目运算符](https://www.baidu.com/s?wd=%E5%8F%8C%E7%9B%AE%E8%BF%90%E7%AE%97%E7%AC%A6&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3nHcvuW6LrHmdnAnvuHR40ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWTknjRsnWm3" \t "http://zhidao.baidu.com/_blank)是运算对象有两个的；如：加(+)减(-)乘(\*)除（/）、自加（++）、自减（--）、逻辑与（||）、逻辑或（&&）、取余（%）、赋值（=）等；  
[三目运算符](https://www.baidu.com/s?wd=%E4%B8%89%E7%9B%AE%E8%BF%90%E7%AE%97%E7%AC%A6&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3nHcvuW6LrHmdnAnvuHR40ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWTknjRsnWm3" \t "http://zhidao.baidu.com/_blank)在Java语言中我知道的就一个（？：）

[左结合运算符](https://www.baidu.com/s?wd=%E4%B8%89%E7%9B%AE%E8%BF%90%E7%AE%97%E7%AC%A6&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3nHcvuW6LrHmdnAnvuHR40ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWTknjRsnWm3" \t "http://zhidao.baidu.com/_blank)在Java语言中所有的二元运算符都是左结合的;

[右结合运算符](https://www.baidu.com/s?wd=%E4%B8%89%E7%9B%AE%E8%BF%90%E7%AE%97%E7%AC%A6&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3nHcvuW6LrHmdnAnvuHR40ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWTknjRsnWm3" \t "http://zhidao.baidu.com/_blank)在Java语言中所有的赋值操作是右结合的.

【例2.81】创建PriorityPractice类，验证一些运算符的优先级；

class PriorityPractice{

public static void main(String[] args){

int a = 20 ;

a = a++ + 1 ;

System.out.println("我们一定要注意运算符的优先级及其在计算机内存中的运算"+a);

a = ++a + 1 ;

System.out.println("我们一定要注意运算符的优先级及其在计算机内存中的运算"+a);

}

}



分析：这个过程分开分析是这样的a=a++----> b=a; a=a+1; a=b;

a=a++ + 1在计算机的执行过程是，1. 备份a+1的值，2. 把a的值加1， 3. 把备份的值再赋值给a

那 a = ++a + 1，就是 1.把a的值加1,2.备份a+1+1的值,3,把a+1+1的值赋值给a