Java的位运算符详解实例

位运算符主要针对二进制，它包括了：“与”、“非”、“或”、“异或”。从表面上看似乎有点像逻辑运算符，但逻辑运算符是针对两个关系运算符来进行逻辑运算，而位运算符主要针对两个二进制数的位进行逻辑运算。下面详细介绍每个位运算符。

1．与运算符

与运算符用符号“&”表示，其使用规律如下：

两个操作数中位都为1，结果才为1，否则结果为0，例如下面的程序段。

```
public class data13
{
public static void main(String[] args)
{
int a=129;
int b=128;
System.out.println("a 和b 与的结果是："+(a&b));
}
}
```

运行结果

```
a 和b 与的结果是：128
```

下面分析这个程序：

“a”的值是129，转换成二进制就是10000001，而“b”的值是128，转换成二进制就是10000000。根据与运算符的运算规律，只有两个位都是1，结果才是1，可以知道结果就是10000000，即128。

2．或运算符

或运算符用符号“|”表示，其运算规律如下：

两个位只要有一个为1，那么结果就是1，否则就为0，下面看一个简单的例子。

```
public class data14
{
public static void main(String[] args)
{
int a=129;
int b=128;
System.out.println("a 和b 或的结果是："+(a|b));
}
}
```

运行结果

```
a 和b 或的结果是：129
```

下面分析这个程序段：
a 的值是129，转换成二进制就是10000001，而b 的值是128，转换成二进制就是10000000，根据或运算符的运算规律，只有两个位有一个是1，结果才是1，可以知道结果就是10000001，即129。

3．非运算符
非运算符用符号“~”表示，其运算规律如下：

如果位为0，结果是1，如果位为1，结果是0，下面看一个简单例子。

```
public class data15
{
public static void main(String[] args)
{
int a=2;
System.out.println("a 非的结果是："+(~a));
}
}
```

4．异或运算符
异或运算符是用符号“^”表示的，其运算规律是：
两个操作数的位中，相同则结果为0，不同则结果为1。下面看一个简单的例子。

```
public class data16
{
public static void main(String[] args)
{
int a=15;
int b=2;
System.out.println("a 与 b 异或的结果是："+(a^b));
}
}
```

运行结果

```
a 与 b 异或的结果是：13
```

分析上面的程序段：a 的值是15，转换成二进制为1111，而b 的值是2，转换成二进制为0010，根据异或的运算规律，可以得出其结果为1101 即13。

Java中的运算符（操作符）

程序的基本功能是处理数据，任何编程语言都有自己的运算符。因为有了运算符，程序员才写出表达式，实现各种运算操作，实现各种逻辑要求。

为实现逻辑和运算要求，编程语言设置了各种不同的运算符，且有优先级顺序，所以有的初学者使用复杂表达式的时候搞不清楚。这里详细介绍一下Java中的运算符。

Java运算符很多，下面按优先顺序列出了各种运算符。

优先级	运算符分类	结合顺序	运算符
由高到低	分隔符	左结合	. [] ( ) ; ,
	一元运算符	右结合	! ++ -- - ~
	算术运算符 移位运算符	左结合	* / % + - << >> >>>
	关系运算符	左结合	< > <= >= instanceof(Java 特有) = = !=
	逻辑运算符	左结合	! && || ~ & | ^
	三目运算符	右结合	布尔表达式?表达式1:表达式2
	赋值运算符	右结合	= *= /= %= += -= <<= >>= >>>= &= *= |=

一、一元运算符

因操作数是一个，故称为一元运算符。

运算符	含义	例子
-	改变数值的符号，取反	-x（-1*x）
~	逐位取反，属于位运算符	~x
++	自加1	x++
--	自减1	x--

++x 因为++在前，所以先加后用。
x++ 因为++在后，所以先用后加。

注意：a+ ++b和a+++b是不一样的（因为有一个空格）。

```
 int a=10;
 int b=10;
 int sum=a+ ++b;
 System.out.println("a="+a+",b="+b+",sum="+sum);
```
运行结果是： a=10,b=11,sum=21

```
 int a=10;
 int b=10;
 int sum=a+++b;
 System.out.println("a="+a+",b="+b+",sum="+sum);
```
运行结果是：a=11,b=10,sum=20

```
n=10;
m=~n;
```
变量n的二进制数形式： 00000000 00000000 00000000 00001010
逐位取反后，等于十进制的-11： 11111111 11111111 11111111 11110101

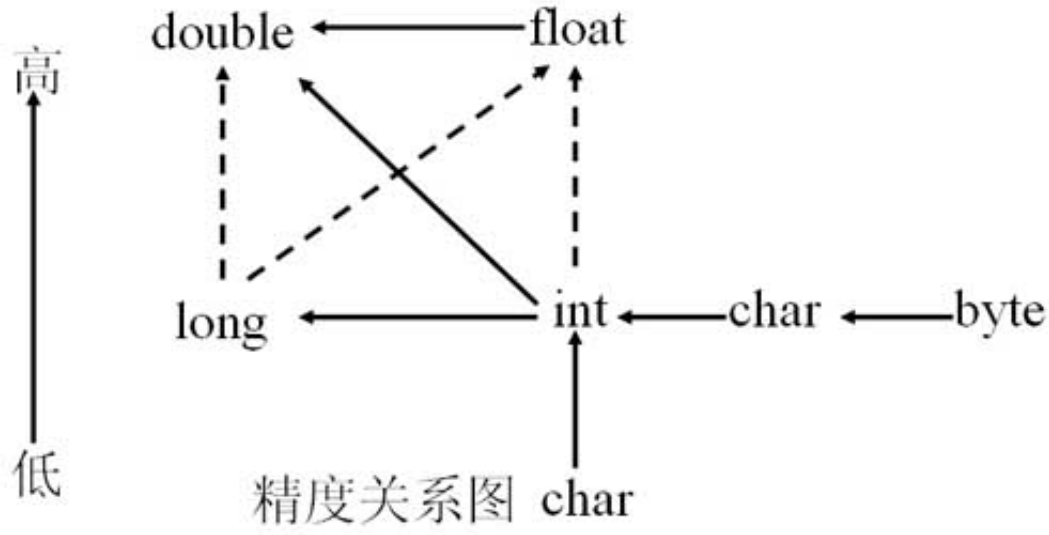
二、算术运算符

所谓算术运算符，就是数学中的加、减、乘、除等运算。因算术运算符是运算两个操作符，故又称为二元运算符。

运算符	含义	例子
+	加法运算	x+y

-	减法运算	x-y
*	乘法运算	x*y
/	除法运算	x/y
%	取模运算（求余运算）	x%y

这些操作可以对不同类型的数字进行混合运算，为了保证操作的精度，系统在运算过程中会做相应的转化。数字精度的问题，我们在这里不再讨论。下图中展示了运算过程中，数据自动向上造型的原则。



注：1、实线箭头表示没有信息丢失的转换，也就是安全性的转换，虚线的箭头表示有精度损失的转化，也就是不安全的。

2、当两个操作数类型不相同时，操作数在运算前会子松向上造型成相同的类型，再进行运算。

示例如下：

[java] view plaincopy

```
1. int a=22;
2. int b=5;
3. double c=5;
```

```
4.
5.  System.out.println(b+"+"+c+"="+(b+c));
6.  System.out.println(b+"-"+c+"="+(b-c));
7.  System.out.println(b+"*"+c+"="+(b*c));
8.  System.out.println(a+"/"+b+"="+(a/b));
9.  System.out.println(a+"%"+b+"="+(a%b));
10. System.out.println(a+"/"+c+"="+(a/c));
11. System.out.println(a+"%"+c+"="+(a%c));
```

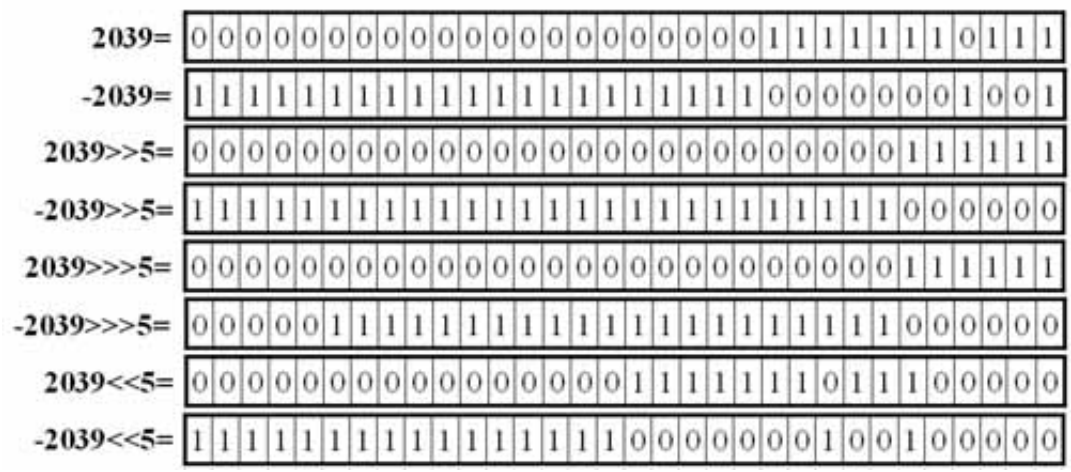
运行结果如下：

```
5+5.0=10.0
5-5.0=0.0
5*5.0=25.0
22/5=4
22%5=2
22/5.0=4.4
22%5.0=2.0
```

三、移位运算符

移位运算符操作的对象就是二进制的位，可以单独用移位运算符来处理int型整数。

运算符	含义	例子
<<	左移运算符，将运算符左边的对象向左移动运算符右边指定的位数（在低位补0）	x<<3
>>	"有符号"右移运算 符，将运算符左边的对象向右移动运算符右边指定的位数。使用符号扩展机制，也就是说，如果值为正，则在高位补0，如果值为负，则在高位补1.	x>>3
>>>	"无符号"右移运算 符，将运算符左边的对象向右移动运算符右边指定的位数。采用0扩展机制，也就是说，无论值的正负，都在高位补0.	x>>>3



以int类型的6297为例，代码如下：

[java] view plaincopy

```
1. System.out.println(Integer.toBinaryString(6297));
2. System.out.println(Integer.toBinaryString(-6297));
3. System.out.println(Integer.toBinaryString(6297>>5));
4. System.out.println(Integer.toBinaryString(-6297>>5));
5. System.out.println(Integer.toBinaryString(6297>>>5));
6. System.out.println(Integer.toBinaryString(-6297>>>5));
7. System.out.println(Integer.toBinaryString(6297<<5));
8. System.out.println(Integer.toBinaryString(-6297<<5));
```

运行结果：

```
1100010011001
11111111111111111110011101100111
11000100
11111111111111111111111100111011
11000100
11111111111111111100111011
110001001100100000
11111111111111001110110011100000
```

注：x<<y 相当于 $x*2^y$ ； x>>y相当于$x/2^y$

从计算速度上讲，移位运算要比算术运算快。

如果x是负数，那么x>>>3没有什么算术意义，只有逻辑意义。

四、关系运算符

Java具有完备的关系运算符，这些关系运算符同数学中的关系运算符是一致的。具体说明如下：

运算符	含义	例子
<	小于	x<y
>	大于	x>y
<=	小于等于	x<=y
>=	大于等于	x>=y
==	等于	x==y
!=	不等于	x!=y

instanceof操作符用于判断一个引用类型所引用的对象是否是一个类的实例。操作符左边的操作元是一个引用类型，右边的操作元是一个类名或者接口，形式如下：

obj instanceof ClassName　　或者　obj instanceof InterfaceName

关系运算符产生的结果都是布尔型的值，一般情况下，在逻辑与控制中会经常使用关系运算符，用于选择控制的分支，实现逻辑要求。

需要注意的是：关系运算符中的"=="和"!="既可以操作基本数据类型，也可以操作引用数据类型。操作引用数据类型时，比较的是引用的内存地址。所以在比较非基本数据类型时，应该使用equals方法。

五、逻辑运算符

逻辑非关系值表

A	!A
true	false
false	true

逻辑与关系值表

A	B	A&&B
false	false	false
true	false	false
false	true	false
true	true	true

逻辑或关系值表

A	B	A||B
false	false	false
true	false	true
false	true	true
true	true	true

在运用逻辑运算符进行相关的操作，就不得不说“短路”现象。代码如下：

if(1==1 && 1==2 && 1==3){ }

代码从左至右执行，执行第一个逻辑表达式后：true && 1==2 && 1==3
执行第二个逻辑表达式后：true && false && 1==3
因为其中有一个表达式的值是false，可以判定整个表达式的值是false，就没有必要执行第三个表达式了，所以java虚拟机不执行1==3代码，就好像被短路掉了。

逻辑或也存在“短路”现象，当执行到有一个表达式的值为true时，整个表达式的值就为true，后面的代码就不执行了。
“短路”现象在多重判断和逻辑处理中非常有用。我们经常这样使用：

[java] view plaincopy

```
1.  public void a(String str){
2.    if(str!=null && str.trim().length()>0){
3.
4.    }
5.  }
```

如果str为null，那么执行str.trim().length()就会报错，短路现象保证了我们的代码能够正确执行。
在书写布尔表达式时，首先处理主要条件，如果主要条件已经不满足，其他条件也就失去了处理的意义。也提高了代码的执行效率。

位运算是对整数的二进制位进行相关操作，详细运算如下：

非位运算值表

A	~A

A	~A
1	0
0	1

与位运算值表

A	B	A&B
1	1	1
1	0	0
0	1	0
0	0	0

或位运算值表

A	B	A | B
1	1	1
1	0	1
0	1	1
0	0	0

异或位运算值表

A	B	A&B
1	1	0
1	0	1
0	1	1
0	0	0

示例如下：

[java] view plaincopy

```
1. int a=15;
2. int b=2;
3.
4. System.out.println(a+"&"+b+"="+(a&b));
5. System.out.println(a+"|"+b+"="+(a|b));
6. System.out.println(a+"^"+b+"="+(a^b));
```

运算结果如下：

15&2=2
15|2=15
15^2=13

程序分析：

a	1	1	1	1	15
b	0	0	1	0	2
a&b	0	0	1	0	2
a|b	1	1	1	1	15
a^b	1	1	0	1	13

按位运算属于计算机低级的运算，现在我们也不频繁的进行这样的低级运算了。

六、三目运算符

三目运算符是一个特殊的运算符，它的语法形式如下：

布尔表达式？表达式1：表达式2

运算过程：如果布尔表达式的值为true，就返回表达式1的值，否则返回表达式2的值，例如：

```
int sum=90;
String str=sum<100 ? "失败" : "成功";
```

等价于下列代码：

```
String str=null;
if(num<100){
  str="失败";
}else{
  str="成功";
}
```

三目运算符和if……else语句相比，前者使程序代码更加简洁。

七、赋值运算符

赋值运算符是程序中最常用的运算符了，示例如下：

运算符	例子	含义
+=	x+=y	x=x+y
-=	x-=y	x=x-y

*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y
>>=	x>>=y	x=x>>y
>>>=	a>>>=y	x=x>>>y
<<=	a<<=y	x=x<<y
&=	x&=y	x=x&y
|=	x|=y	x=x|y
^=	x^=y	x=x^y

大家可以根据自己的喜好选择合适的运算符。

补充：

字符串运算符： + 可以连接不同的字符串。

转型运算符： () 可以将一种类型的数据或对象，强制转变成另一种类型。如果类型不相容，会报异常出来。

[java] view plain copy

```
package com.zf.binary;

public class Test1 {

    public static void main(String[] args) {

        /* 符号为:最高位同时表示图号，0为正数，1为负数 */

        /*
            1、二进制转换为十进制

            二进制转换为10进制的规律为： 每位的值 * 2的（当前位-1次方）
            例如：
                00000001 = 0 * 2^7 + 0 * 2^6 + 0 * 2^5 + 0 * 2^4 + 0 * 2^3 + 0 * 2
^2 + 0 * 2^1 + 1 * 2^0  = 1
```

```
                00000010 = 0 * 2^7 + 0 * 2^6 + 0 * 2^5 + 0 * 2^4 + 0 * 2^3 + 0 * 2
^2 + 1 * 2^1 + 0 * 2^0  = 2

            2、二进制的符号位：
                最高位表示符号位，0表示正数  ，  1表示负数


            3、将二进制负数转换为十进制：先对该二进制数取反，然后加1，再转换为十进制，然
后在前面加上负号
                例如：  10101011 最高位为1，所以为负数
                   第一步：取反：  01010100
                   第二步：加1 ：  01010101
                   第三步：转换为10进制：85
                   第四步：加上负号：  -85
                   所以       10101011 转换为十进制为 -85

            4、将十进制负数转换为二进制：先得到该十进制负数的绝对值，然后转换为二进制，然
后将该二进制取反，然后加1
                例如：-85
                第一步：得到绝对值 85
                第二步：转换为二进制：01010101
                第二步：取反：                  10101010
                第三步：加1：                   10101011
                所以，-85转换为二进制为  10101011
         */

        /*
        ~ ‘非’运算符是将目标数的进制去反，即0变成1 ，1变成0
        2的二进制码为 00000010 ，  它取反为11111101 ，可见取反后结果为负数（二进制负数转
换为十进制的步骤为：将二进制去反，然后+1）
         将 11111101 转换为10进制 ，第一步去反 得到 00000010 然后 加1 得到 00000011 ，
得到的结果为3 ，然后在前面加上负号就可以了
         所以结果为-3
         */
        System.out.println(~2);

        /*
          ^ 异或 ，计算方式为：两个二进制数的位相同则为0 不同则为1
          23转换为二进制为：00010111
          12转换为二进制为：00001100
                    计算结果为：00011011  =  27
         */
        System.out.println(23 ^ 12);

        /*
         & 按位与 ，计算方式为：两个二进制数的位都为1则为1 ，否则为0
         1的二进制为 ：00000001
         2的二进制为 ：00000010
              结果为 :00000000 = 0
         */
        System.out.println(1&2);
```

```
        /*
          | 按位或 ，计算方式为：两个二进制位有一个为1就为1，否者为0
          5 的二进制为：00000101
          6 的二进制为：00000110
                结果为：00000111 = 7
         */
        System.out.println( 5 | 6);


        /*
          >> 有符号右移位  ，符号左边表示要被移位的数，右边表示需要移的位数，结果为正数则在左边补0，否则补1
          3 的二进制为：00000010
                向右移动1位：00000001 = 1
         */
        System.out.println(3 >> 1);


    }

}
```