常用的位运算符，&、|、~、^、<<、>>、>>>，java 版本

作者：leoxcyu
原文链接：https://ld246.com/article/1653921612902
来源网站：链滴

位运算符，&、|、~、^、<<、>>、>>>

位运算符主要针对二进制，它包括了：” 与 “、” 或 “、” 非 “、” 异或 “，与逻辑运算符符号相识但逻辑运算符是针对两个关系来进行逻辑运算。

1.与运算符，符号：&，

==使用规律：两个操作数中，对应位都为1，结果才为1，否则结果为0==

```
public static void andOperator() {
    int a = 129, b = 128;
    String aBinaryCode = Integer.toString(a, 2);
    String bBinaryCode = Integer.toString(b, 2);
    System.out.println("a的二进制表示为：" + aBinaryCode);
    System.out.println("b的二进制表示为：" + bBinaryCode);
    System.out.println("a&b二进制表示为: " + Integer.toString((a & b), 2));
    System.out.println("a&b的结果是: " + (a & b));
}
```

2.或运算符，符号：|

==使用规律：两个操作数中，对应位只要有一个为1，结果才为1，否则结果为0==

```
public static void orOperator() {
    int a = 129, b = 128;
    String aBinaryCode = Integer.toString(a, 2);
    String bBinaryCode = Integer.toString(b, 2);
    System.out.println("a的二进制表示为：" + aBinaryCode);
    System.out.println("b的二进制表示为：" + bBinaryCode);
    System.out.println("a|b二进制表示为: " + Integer.toString((a | b), 2));
    System.out.println("a|b的结果是: " + (a | b));
}
```

3.非运算符，符号：~

==使用规律：如果位为0，结果为1、如果位为1，结果为0，按位取反==

```
public static void notOperator() {
    int a = -37;
    String aBinaryCode = Integer.toString(a, 2);
    System.out.println("a的二进制表示为：" + aBinaryCode);
    System.out.println("~a二进制表示为: " + Integer.toString(~a, 2));
    System.out.println("~a的结果是: " + ~a);

    /**
     * int 长度为8个字节，一个字节8位，占32位
     * 以37为例：转为二进制：100101，
     * 补码为：00000000 00000000 00000000 00100101
     * 取反为：11111111 11111111 11111111 11011010
     * 因为最高为是1，所以原码为负数，负数的补码是其绝对值的原码取反，末尾再加1
     * 所以将补码还原 ，高位不变，是减1，再取反  减一后为：11111111 11111111 11111111 1
```

```
011001
     *                    取反后为：10000000 00000000 00000000 00100110
     *                    为-38
     * 个人总结（不一定对，仅供参考）：~运算符，为将原来的值+1，再加-号(负号)
     * eg:~36=-37、~(-36)=35
     */
  }
```

4.异或运算符，符号：^

==使用规律：两个操作数中，对应位相同，结果才为0，否则结果为1==

```
  public static void xorOperator() {
    int a = 129, b = 128;
    String aBinaryCode = Integer.toString(a, 2);
    String bBinaryCode = Integer.toString(b, 2);
    System.out.println("a的二进制表示为：" + aBinaryCode);
    System.out.println("b的二进制表示为：" + bBinaryCode);
    System.out.println("a^b二进制表示为: " + Integer.toString((a ^ b), 2));
    System.out.println("a^b的结果是: " + (a ^ b));
  }
```

5.左移运算符,符号:<<

==使用规律：左操作数，按为左移右操作数指定位数(后面直接补0)==

eg:2 << 3=2*Math.pow(2,3)

```
  public static void moveLeft() {
    int a = 2;
    String aBinaryCode = Integer.toString(a, 2);
    System.out.println("a的二进制表示为：" + aBinaryCode);
    System.out.println("a<<3二进制表示为: " + Integer.toString(a << 3, 2));
    System.out.println("a<<3的结果是: " + (a << 3));
  }
```

6.右移运算符,符号：>>

==使用规律：左操作数，按为右移右操作数指定位数 eg:8 >> 3=8/Math.pow(2,3)==

如果该数为正，则高位补0，而若该数为负数，则右移后高位补1，以上特性不变

```
  public static void moveRight() {
    int a = 8;
    String aBinaryCode = Integer.toString(a, 2);
    System.out.println("a的二进制表示为：" + aBinaryCode);
    System.out.println("a>>3二进制表示为: " + Integer.toString(a >> 3, 2));
    System.out.println("a>>3的结果是: " + (a >> 3));

  }
```

7.无符号右移运算符,符号：>>>

==也叫逻辑右移，即该数为正，则高位补0，而若该数为负数，则右移后高位同样补0==

```
    public static void moveLogicRight() {

        //a为正数时，>> 和 >>>没有区别,正数的原码，补码，反码基本不变
        int a = 8;
        String aBinaryCode = Integer.toString(a, 2);
        System.out.println("a的二进制表示为：" + aBinaryCode);
        System.out.println("a>>>3二进制表示为: " + Integer.toString(a >>> 3, 2));
        System.out.println("a>>>3的结果是: " + (a >>> 3));

        //当a为负数时
        int b = -8;
        String bBinaryCode = Integer.toString(b, 2);
        System.out.println("b的二进制表示为：" + bBinaryCode);
        System.out.println("b>>>3二进制表示为: " + Integer.toString(b >>> 3, 2));
        System.out.println("b>>>3的结果是: " + (b >>> 3));

        /**
         * 负数的二进制为其正数的补码+1,第一位改为1 本例: b=-8
         * 原码：最高位为1 ,其余位为其正数的二进制 : 10000000 00000000 00000000 00001000
         * 反码：按位取反，11111111 11111111 11111111 11110111
         * 补码：最高位不变，其它位按位取反 然后+1 （针对原码而言），
         * 实际就是反码+1 ，注意逢2进1 :11111111 11111111 11111111 11111000
         * 右移三位：高位补0：    00011111 11111111 11111111 11111111
         * 位运算都是对补码左右移
         * System.out.println(Integer.parseInt("00011111111111111111111111111111",2));
         */

    }
```