Java7.14 位运算符(BitOperator)

今天是7.14 这章节和原码反码补码是重点 要灵活学习和运用

1. 位运算符详细学习记录1

<java7.14原码反码补码及位运算题目思考.docx>

Java中有7个位运算 分别是（& | ^ ~ >> << >>>）

按位与& : 两位全为1 结果为1 否则为0

按位或| : 两位有一个为1 结果为1 否则为0

按位异或^ : 两位一个为0 一个为1 结果为1 否则为0

按位取反~ : 0->1 1->0

比如： 2&3=? ~-2=? ~2=? 2|3=? 2^3=?

下面是我在idea中运行所得（带详细解析过程）：

这是求2&3

public class BitOperatorExercise1 {  
 public static void main(String[]args){  
 //详细解析位运算符  
 //解析： 因为在计算机运行的时候 都是以补码的方式来运行的 所以2&3 先求出2和3的补码  
 //正数的三码合一（原码补码反码相同 可以看7.14word里面有写）  
 //第一步 求出2和3的补码 （int类型四个字节） 正数的符号位是0 负数的符号位是1  
 //2的补码 00000000 00000000 00000000 00000010  
 //3的补码 00000000 00000000 00000000 00000011  
 //第二步 求2&3的补码 按位与（两者都为一 结果为1 否则为0）  
 //2的补码 00000000 00000000 00000000 00000010  
 //3的补码 00000000 00000000 00000000 00000011  
 //2&3的补码 00000000 00000000 00000000 00000010  
 //第三步 2&3的补码已得到 但是在计算机中是以原码为展示结果的（原码反码补码知识见7.14word）  
 //2&3的原码 00000000 00000000 00000000 00000010  
 //所以输出结果为2  
 System.*out*.println(2&3);//输出为2  
 }  
}

这里是求~-2

//解析(~-2)  
//都一步 求出-2的补码 （负数的补码等于反码+1 7.14原码反码补码word有写）  
//-2的补码等于反码+1 负数的反码等于原码的符号位不变 其他位取反  
//-2的原码为 10000000 00000000 00000000 00000010  
//-2的反码为 11111111 11111111 11111111 11111101  
//-2的补码为 11111111 11111111 11111111 11111110  
//第二步 求~-2 ~按位取反 0->1 1->0  
//-2的补码为 11111111 11111111 11111111 11111110  
//~-2的补码为 00000000 00000000 00000000 00000001  
//第三步 展示结果为原码 求~-2的原码 ~-2是个正数 正数三码合一  
//即~-2的原码为 00000000 00000000 00000000 00000001  
//输出结果为1  
System.*out*.println(~-2);//输出为1

这里是求~2 按位取反

//解析(~2)  
//第一步求出2的补码  
//2的补码为 00000000 00000000 00000000 00000010  
//第二步求~2 ~按位取反 0->1 1->0  
//2的补码为 00000000 00000000 00000000 00000010  
//~2的补码为 11111111 11111111 11111111 11111101  
//第三步 求~2的原码  
//首先~2是个负数 它的补码等于反码+1 而它的反码等于原码的符号位不变 其他取反  
//~2的反码为补码-1 即11111111 11111111 11111111 11111100  
//~2的原码为 10000000 00000000 00000000 00000011  
//第四步输出 即为-3 （0 1 10 11 100 101 110）  
System.*out*.println(~2);//输出为-3

这里是求2|3=? 2^3=?

//解析2|3  
//第一步求出2和3的补码  
//2的补码 00000000 00000000 00000000 00000010  
//3的补码 00000000 00000000 00000000 00000011  
//第二步求2|3 两位有一个为1 结果为1 否为为0  
//2|3的补码 00000000 00000000 00000000 00000010  
//第三步求出2|3的补码输出  
//即为3  
System.*out*.println(2|3);//输出为3  
  
//解析2^3  
//第一步求出2和3的补码  
//2的补码 00000000 00000000 00000000 00000010  
//3的补码 00000000 00000000 00000000 00000011  
//第二步 求2^3 两位一个为0 一个为1 结果为1 否则为0  
//2^3的补码 00000000 00000000 00000000 00000001  
//第三步 输出2^3的原码 00000000 00000000 00000000 00000001  
//即为1  
System.*out*.println(2^3);//输出为1

1. 位运算符详细学习记录2

>> << >>>运算规则

1. 算术右移>> 低位溢出 符号位不变 并用符号位补溢出的高位
2. 算术左移<< 符号位不变 低位补0
3. 逻辑右移（也叫无符号右移）>>> 低位溢出，高位补 0
4. 特别说明 没有<<<符号

比如：

int a = 1 >> 2; => 00000000 00000000 00000000 00000001

* 00000000 00000000 00000000 00000000 (01)
* 本质：1 / 2 / 2 = 0

int a = 12 >> 3; => 00000000 00000000 00000000 00000001

* 00000000 00000000 00000000 00000000
* 本质：12 / 2 / 2 / 2 = 1

int b = 1 << 2; => 00000000 00000000 00000000 00000001

* 00000000 00000000 00000000 00000100
* 本质：1 \* 2 \* 2 = 4

int b = 4 << 3; => 00000000 00000000 00000000 00000001

* 本质：4 \* 2 \* 2 \* 2 = 32

int c = 1 >>> 2; => 00000000 00000000 00000000 00000001

* 00000000 00000000 00000000 00000000 (01)

public class BitOperatorExercise2 {  
 public static void main(String[]args){  
 System.*out*.println(8 >> 2);//本质 8 / 2 / 2 = 2  
 System.*out*.println(4 >>> 2);//本质 4 / 2 / 2 = 1  
 System.*out*.println(24 >>> 3);//本质 24 / 2 / 2 / 2 = 3  
 System.*out*.println(24 >> 3);//本质 24 / 2 / 2 / 2 = 3  
 System.*out*.println(2 << 3);//本质 2 \* 2 \* 2 \* 2 =16  
 }  
}