Java 位运算

作者：DongXiaokai0819
原文链接：https://ld246.com/article/1573105372292
来源网站：链滴

位运算

1、原码, 反码, 补码

    对于一个数，计算机
使用一定地编码方式进行存储，原码、反码、补码是机器存储一个具体数字的编码方式。

原码，反码，补码的产生过程，就是为了解决，计算机做减法和引入符号位（正号和负号）的问题。

1.1 机器数

● 一个数在在计算机中的二进制表示形式，叫做这个数的机器数。机器数是带符号的，在计算机用一数的最高位存放符号, 正数为0, 负数为1.

    比如，十进制中的数 +
，计算机字长为8位，转换成二进制就是00000011。如果是 -3 ，就是 10000011 。

那么，这里的 00000011 和 10000011 就是机器数。

1.2 真值

    因为第一位是符号位
所以机器数的形式值就不等于真正的数值。例如上面的有符号数 10000011，其最高位1代表负，其正数值是 -3 而不是形式值131（10000011转换成十进制等于131）。

● 所以，为区别起见，将带符号位的机器数对应的真正数值称为机器数的真值。

例：0000 0001的真值 = +000 0001 = +1，1000 0001的真值 = –000 0001 = –1

1.3 原码

● 原码是人脑最容易理解和计算的表示方式。

    原码就是符号位加上
值的绝对值, 即用第一位表示符号, 其余位表示值.

比如如果是8位二进制:

● [+1]原 = 0000 0001

● [-1]原 = 1000 0001

因为第一位是符号位，所以8位二进制的取值范围是:

[1111 1111 , 0111 1111] 即 [-127 , 127]

1.4 反码

反码的表示方法是:

● 正数的反码是其本身。

- 负数的反码是在其原码的基础上, 符号位不变，其余各个位取反。

示例:

- [+1] = [00000001]原 = [00000001]反
- [-1] = [10000001]原 = [11111110]反

如果一个反码表示的是负数, 人脑无法直观的看出来它的数值. 通常要将其转换成原码再计算.

1.5 补码

补码的表示方法是:

- 正数的补码就是其本身。
- 负数的补码是在其原码的基础上, 符号位不变, 其余各位取反, 最后+1. (即在反码的基础上+1)。

负数补码的另一种算法:

负数的补码等于他的原码自低位向高位，尾数的第一个‘1’及其右边的‘0’保持不变，左边的各位位取反，符号位不变。

示例:

- [+1] = [00000001]原 = [00000001]反 = [00000001]补
- [-1] = [10000001]原 = [11111110]反 = [11111111]补

对于负数, 补码表示方式也是人脑无法直观看出其数值的. 通常也需要转换成原码在计算其数值。

1.6 负数

    现在我们知道了计算
可以有三种编码方式表示一个数. 对于正数因为三种编码方式的结果都相同:

- [+1] = [00000001]原 = [00000001]反 = [00000001]补

所以不需要过多解释. 但是对于负数:

- [-1] = [10000001]原 = [11111110]反 = [11111111]补

1.7 参考及详解传送门:

- https://blog.csdn.net/zl10086111/article/details/80907428
- https://blog.csdn.net/zhiwen_a/article/details/81192087

2、位运算

    位运算主要包括按位与
&)、按位或(|)、按位异或(^)、取反()、左移(<<)、右移(>>)这几种。
    其中除了取反(

以外，其他的都是二目运算符，即要求运算符左右两侧均有一个运算量。

2.1 按位与(&)

    参加运算的两个数，
算为二进制后，进行与（&）运算。只有当相应位上的数都是1时，该位才取1，否则该为为0。

运算规则：两位同时为“1”，结果才为“1”，否则为0

- 0&0=0;
- 0&1=0;
- 1&0=0;
- 1&1=1;

示例：

3&5 即 0000 0011& 0000 0101 = 00000001 因此，3&5的&运算值为1。

另，负数按补码形式参加按位或运算。

与(&)运算的用途

1. 清零。如果想将一个单元清零，即使其全部二进制位为0，只要与一个各位都为零的数值相与，结为零。
2. 取一个数中指定位。（00001111 即取一个数的低四位）

2.2 按位或(|)

    参加运算的两个数，
算为二进制后，进行或（|）运算。只要相应位上存在1，那么该位就取1，均不为1，即为0。

运算规则：参加运算的两个对象只要有一个为1，其值为1。

- 0|0=0;
- 0|1=1;
- 1|0=1;
- 1|1=1;

示例：

3|5 即 00000011 | 0000 0101 = 00000111 因此，3|5的|运算的值为7。

另，负数按补码形式参加按位或运算。

或(|)运算的用途

1. 常用来对一个数据的某些位置 置为1。

例：将X=10100000的低4位置1，用X | 0000 1111 = 1010 1111即可得到。

2.3 按位异或(^)

    参加运算的两个数，
算为二进制后，进行异或运算。只有当相应位上的数字不相同时，该为才取1，若相同，即为0。

运算规则：参加运算的两个对象，如果两个相应位为“异”（值不同），则该位结果为1，否则为0。

- 0^0=0;
- 0^1=1;
- 1^0=1;
- 1^1=0;

示例：

10 ^ -10 即 0000 1010 ^ 1111 0110 = 1111 1100

异或(^)的用途

异或其实就是不进位加法,如 1+1=0 ,0+0=0 , 1+0=1。

1. 使特定位翻转找一个数，对应X要翻转的各位，该数的对应位为1，其余位为零，此数与X对应位异即可。

例：X=10101110，使X低4位翻转，用X ^0000 1111 = 1010 0001即可得到。

2. 与0相异或，保留原值 ， X ^ 00000000 = 1010 1110。

2.4 取反(~)

    参加运算的两个数，
算为二进制后，进行取反运算。每个位上都取相反值，1变成0，0变成1。

运算规则：单目运算符，1变0，0变1。

- ~0=1;
- ~1=0;

2.5 左移(<<)

    参加运算的两个数，
算为二进制后，进行左移运算，用来将一个数各二进制位全部向左移动若干位。

运算规则：将一个运算对象的各二进制位全部左移若干位（左边的二进制位丢弃，右边补0）。

- 0000 1010<<2 = 0010 1000

若左移时舍弃的高位不包含1，则每左移一位，相当于该数乘以2。

2.6 右移(>>)

    参加运算的两个数，

算为二进制后，进行右移运算，用来将一个数各二进制位全部向右移动若干位。

运算规则：将一个数的各二进制位全部右移若干位，正数左补0，负数左补1，右边丢弃。

- 0000 1010>>2 = 0000 0010

操作数每右移一位，相当于该数除以2。

2.7 参考及详解传送门：

- https://blog.csdn.net/sinat_35121480/article/details/53510793
- https://blogread.cn//it/article/7327?f=wb

3、Java中的位运算

- 以下内容转自：　https://www.jianshu.com/p/416dd46dea7b

    Java 中的位运算共有
(&)、或(|)、非(~)、异或(^)、左移(<<)、右移(>>)、无符号右移(>>>) 这7种，概览见下表：

符号	含义	详解
&	位与	两个比特位都为 1 时，结果才为 1，否则为 0 （位与操作满足交换律和结合律，甚至分配律）
|	位或	两个比特位都为 0 时，结果才为 0，否则为 1 （位或操作满足交换律和结合律，甚至分配律）
~	位非	即按位取反，1 变 0，0 变 1
^	异或	两个比特位相同时（都为 0 或都为 1）为 0，相异为 1（异或操作满足交换律和结合律，甚至分配律。任何整数和自己异或的结果为 0，任何整数与 0 异或值不变）
<<	左移	将所有的二进制位按值向←左移动若干位，左移操作与正负数无关，它只是傻傻地将所有位按值向左移动，高位舍弃，低位补 0
>>	右移	将所有的二进制位按值向右→移动若干位，低位直接舍弃，跟正负无关，而高位补 0 还是补 1 则跟被操作整数的正负相关，正数补 0 ，负数补 1
>>>	无符号右移	将所有的二进制位按值向右→移动若干位，低位直接舍弃，跟正负无关，高位补 0 ，也跟正负无关

3.1Java位运算基础操作

a|0 == a;

原文链接：Java 位运算

```
a&-1 == a;
a&0 == 0;

a^a == 0;
a^0 == a;

a|~a == -1;
a&~a == 0;
a&a == a;
a|a == a;

a|(a&b) == a;
a&(a|b) == a;
```

3.2位运算进阶操作

a. 判断奇偶

- 可以很简单归纳出，只需判断一个正整数的二进制补码最后一位是0是1，是0为偶数，是1为奇数

```
public void isOddOrEven(int n){
        if ((n & 1) == 1) {//n是奇数
            System.out.println("Odd");
        }else {//n是偶数
            System.out.println("Even");
        }
    }
```

b. 省去中间变量交换两整数的值（面试常考）

```
public void swap(){
        int a = 1, b = 2;
        a ^= b;
        b ^= a;//b == 1
        a ^= b;//a == 2
        System.out.println("a:" + a);//a:2
        System.out.println("b:" + b);//b:1
    }
```

这里还有另一种做法，今早上跟舍友讨论了一下：

```
a = 1,b = 2;
若要让ab俩值交换，则先让a=a+b, 然后 b=a,然后a = a-b;
感觉和位运算差不多的。
```

c. 变换符号，正变负，负变正

- 只需对待操作数应用取反操作后再加 1 即可

```
public void negate(){
        int a = -10, b = 10;
        System.out.println(~a + 1);//10
```

```
        System.out.println(~b + 1);//-10
    }
```

d. 求绝对值，

- 对于负数可以通过上面变换符号的操作得到绝对值，
- 正数直接返回即可，因此我们要先判断符号位来得知当前数的正负。

```
public int abs(int a){
        int i = a >> 31;//得到符号位，0 为正数，-1 为负数
        return i == 0 ? a : (~a + 1);//符号位为 0 直接返回，否则返回 ~a + 1
    }
```

    或者，n>>31 取得n
符号，若n为正数，n>>31等于0，若n为负数，n>>31等于-1。若n为正数 n^0-0 数不变，若n为负 n^-1 需要计算n和-1的补码，异或后再取补码，结果n变号并且绝对值减1，再减去-1就是绝对值

```
public int abs(int a){
        return a ^ (a >> 31)) - (a >> 31);
    }
```

e. 判断一个数num是不是 2 的幂

如果是2的幂，n一定是10000...（也就是二进制位里只有一个是1，且是首位，其余全是0），n-1就是111111... 所以做与运算结果为 0。

```
    public static boolean judgeNum(int num){
        if (num < 1)
            return false;
        return (num & (num - 1)) == 0;
    }
```

f. 判断一个数num是不是 4 的幂

理论上数字4幂的二进制类似于100，10000，1000000，... 这些形式。

不难归纳出如下结论：

4的幂一定是2的。

4的幂和2的幂一样，二进制位里只有一个是1，且是首位。但是，4的幂中的1总是出现在奇数位。

这里贴下 Kotlin 代码（非最佳实现）：

```
    public static boolean judgeNum(int num){
        if (num < 1)
            return false;
        return (num & (num - 1)) == 0 && (Integer.toBinaryString(num).length() & 1) == 1;
    }
```

g. 其他进阶操作

```
// 7. int 型最大值是什么?
```

```
System.out.println((1 << 31) - 1);// 2147483647， 注意运算符优先级，括号不可省略
System.out.println(~(1 << 31));// 2147483647

// 8. int 型最小值是什么?
System.out.println(1 << 31);
System.out.println(1 << -1);

// 9. long 型最大值是什么?
System.out.println(((long)1 << 127) - 1);

// 10. 整数n乘以2是多少?
n << 1;

// 11. 整数n除以2是多少？(负奇数的运算不可用)
n >> 1;

// 12. 乘以2的n次方，例如计算10 * 8(8是2的3次方)
System.out.println(10<<3);

// 13. 除以2的n次方，例如计算16 / 8(8是2的3次方)
System.out.println(16>>3);

// 14. 取两个数的最大值（某些机器上，效率比a>b ? a:b高)
System.out.println(b&((a-b)>>31) | a&(~(a-b)>>31));

// 15. 取两个数的最小值（某些机器上，效率比a>b ? b:a高)
System.out.println(a&((a-b)>>31) | b&(~(a-b)>>31));

// 16. 判断符号是否相同(true 表示 x和y有相同的符号， false表示x，y有相反的符号。)
System.out.println((a ^ b) > 0);

// 17. 计算2的n次方 n > 0
System.out.println(2<<(n-1));

// 18. 求两个整数的平均值
System.out.println((a+b) >> 1);

// 19. 从低位到高位,取n的第m位
int m = 2;
System.out.println((n >> (m-1)) & 1);

// 20. 从低位到高位.将n的第m位置为1，将1左移m-1位找到第m位，
//得000...1...000，n 再和这个数做或运算
System.out.println(n | (1<<(m-1)));

// 21. 从低位到高位,将n的第m位置为0，将1左移m-1位找到第m位，
//取反后变成111...0...1111，n再和这个数做与运算
System.out.println(n & ~(0<<(m-1)));

//22.取模的操作 a % (2^n) 等价于 a&(2^n-1),后者效率更高一点
System.out.println("the 345 % 16 is "+(345%16)+ " or "+(345&(16-1)));
```

3.3 位运算的应用

位运算作为底层的基本运算操作，往往是和'高效'二字沾边，适当的运用位运算来优化系统的核心代，会让你的代码变得十分的精妙。

位运算常用来用作标志位：如1100 ，四位可表示四种标志，1为true，0为false。

下面用一个例子来演示一下。

```
public class Permission {

    // 是否允许查询，二进制第1位，0表示否，1表示是
    public static final int ALLOW_SELECT = 1 << 0; // 0001  = 1

    // 是否允许新增，二进制第2位，0表示否，1表示是
    public static final int ALLOW_INSERT = 1 << 1; // 0010  = 2

    // 是否允许修改，二进制第3位，0表示否，1表示是
    public static final int ALLOW_UPDATE = 1 << 2; // 0100  =4

    // 是否允许删除，二进制第4位，0表示否，1表示是
    public static final int ALLOW_DELETE = 1 << 3; // 1000  = 8

    // 存储目前的权限状态
    private int flag;

    //设置用户的权限
    public void setPer(int per) {
        flag = per;
    }

    //增加用户的权限（1个或者多个）
    public void enable(int per) {
        flag = flag|per;
    }

    //删除用户的权限（1个或者多个）
    public void disable(int per) {
        flag = flag&~per;
    }

    //判断用户的权限
    public boolean isAllow(int per) {
        return ((flag&per)== per);
    }

    //判断用户没有的权限
    public boolean isNotAllow(int per) {
        return ((flag&per)==0);
    }
    public static void main(String[] args) {
        int flag = 15;
        Permission permission = new Permission();
        permission.setPer(flag);
        permission.disable(ALLOW_DELETE|ALLOW_INSERT);
        System.out.println("select = "+permission.isAllow(ALLOW_SELECT));
        System.out.println("update = "+permission.isAllow(ALLOW_UPDATE));
```

原文链接：Java 位运算

```
        System.out.println("insert = "+permission.isAllow(ALLOW_INSERT));
        System.out.println("delete = "+permission.isAllow(ALLOW_DELETE));
    }
}
```

输出如下：

```
select = true
update = true
insert = false
delete = false
```