java的位运算

在学习位运算之前，我们先来了解一下二进制码相关知识。

1，原码

正数的原码就是它本身，10：0000 1010

负数用最高位是1表示负数，-10：1000 1010

2，反码

正数的反码和原码一样，10反码：0000 1010

负数的反码是负数的原码按位取反，符号位不变 -10反码：1111 0101

3，补码 （整数的补码才是计算机中的存储形式）

正数的补码和原码一样

负数的补码是负数的反码加1，-10:1111 0110

4，在计算机种，为什么不用原码和反码，而是用补码呢？

使用原码，反码在计算时不准确，使用补码计算时才准确

了解二进制码规则之后，我们来看位运算

java中包含了七种位运算符

>> 右移，<< 左移，>>> 无符号右移，& 与运算符，布尔代数，

| 或运算符，^ 异或，~ 非运算。

使用场景

1.判断奇偶性

public void method1(int a){

if(a&1==0){

log.info("偶数");

}else if(a&1 ==1){

log.info("奇数");

}

}

2.不使用中间变量交换两个数

public void swap(int a,int b){

a=a^b;

b=b^a;

a=a^b;

}

知识点:任何数和自己进行异或操作结果都为0，异或符合交换律，即 a^b =b^a

上面操作就等于：

a = a ^ b;

b = b ^ a = b ^ (a ^ b) = a;

a = a ^ b = (a ^ b) ^ (b ^ (a ^ b)) = (a ^ b) ^ a = b;

3.判断一个正整数是不是2的整数次幂

public boolean power2(int a) {

if (a <= 0){

System.out.println("这里不计算负数，直接返回false");

return false;

} else{

return (a&(a-1))==0

}

}

**大于零的2的幂数和自己减一后的数进行与运算结果必然为0**

**4.找出没有重复的数**

给你一组整型数据，这些数据中，其中有一个数只出现了一次，其他的数都出现了两次，让你来找出一个数 。

很多人会用一个哈希表来存储，每次存储的时候，记录 某个数出现的次数，最后再遍历哈希表，看看哪个数只出现了一次。这种方法的时间复杂度为 O(n)，空间复杂度也为 O(n)了。

然而我想告诉你的是，采用位运算来做，绝对高逼格！

我们刚才说过，两个相同的数异或的结果是 0，一个数和 0 异或的结果是它本身，所以我们把这一组整型全部异或一下，例如这组数据是：1， 2， 3， 4， 5， 1， 2， 3， 4。其中 5 只出现了一次，其他都出现了两次，把他们全部异或一下，结果如下：

由于异或支持交换律和结合律，所以:

123451234 = （11)(22)(33)(44)5= 00005 = 5。

也就是说，那些出现了两次的数异或之后会变成0，那个出现一次的数，和 0 异或之后就等于它本身。就问这个解法牛不牛逼？所以代码如下

int find(int[] arr){

int tmp = arr[0];

for(int i = 1;i < arr.length; i++){

tmp = tmp ^ arr[i];

}

return tmp;

}

5.m的n次方

labuladong