二进制中的原码、反码、补码

有符号数：

> 对于有符号数而言，符号的正、负机器是无法识别的，但由于“正、负”恰好是两种截然不同的状态，如果用“0”表示“正”，用“1”表示“负”，这样符号也被数字化了，并且规定将它放在有效数字的前面，即组成了有符号数。所以，在二进制中使用最高位（第一位）来表示符号，最高位是0，表示正数；最高位是1，表示负数。

`10000000000000001111100`

无符号数：

> 无符号数是针对二进制来讲的，无符号数的表数范围是非负数。全部二进制均代表数值（所有位都用于表示数的大小），没有符号位。即第一个"0"或"1"不表示正负

`00000000000000001111100`

对于有符号数而言的性质：

> (1)二进制的最高位是符号位：0表示正数，1表示负数
>
> (2)正数的原码、反码、补码都一样
>
> (3)负数的反码 = 它的原码符号位不变，其他位取反（0 ->1 ; 1->0 ）
>
> (4)负数的补码 = 它的反码 +1
>
> (5)0的反码、补码都是0
>
> (6)在计算机运算的时候，都是以补码的方式来运算的

有符号数运算案例

1. 正数相加：

   例如：1+1 ， 在计算机中运算如下：

1的原码为：`00000000 00000000 00000000 00000001`

反码：`00000000 00000000 00000000 00000001`

补码：`00000000 00000000 00000000 00000001`

两数的补码相加：`00000000 00000000 00000000 00000010`（转换为10进制）= 2

2. 正数相减：

例如：1 - 2，在计算机中运算如下：

在计算机中减运算其实是作为加运算来操作的，所以，1-2 = 1 + (-2)

- 第一步：获取1的补码 `00000000 00000000 00000000 00000001`
- 第二步：获取-2的补码

  -2的原码：`10000000 00000000 00000000 00000010`

  -2的反码：`11111111 11111111 11111111 11111101`

  -2的补码：`11111111 11111111 11111111 11111110`

- 第三步：1的补码与-2的补码相加：

  `00000000 00000000 00000000 00000001`

  + `11111111 11111111 11111111 11111110`

  = `11111111 11111111 11111111 11111111`

- 第四步：将计算结果的补码转换为原码，反其道而行之即可（如果想将二进制转换为十进制，必须得到二进制的原码）

  补码：`11111111 11111111 11111111 11111111`

  =

  反码：`11111111 11111111 11111111 11111110`

  =

  原码：`10000000 00000000 00000000 00000001`

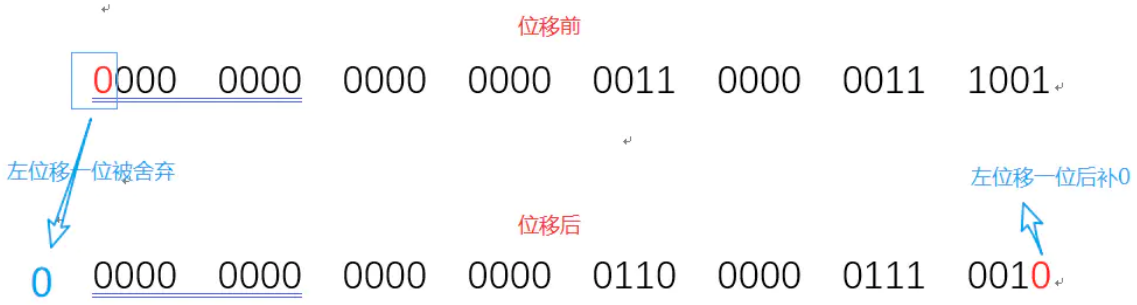
第五步：将计算结果的二进制原码 转换 为十进制

二进制原码：`10000000 00000000 00000000 00000001` = -1

<< 、>>、>>> 位移运算符

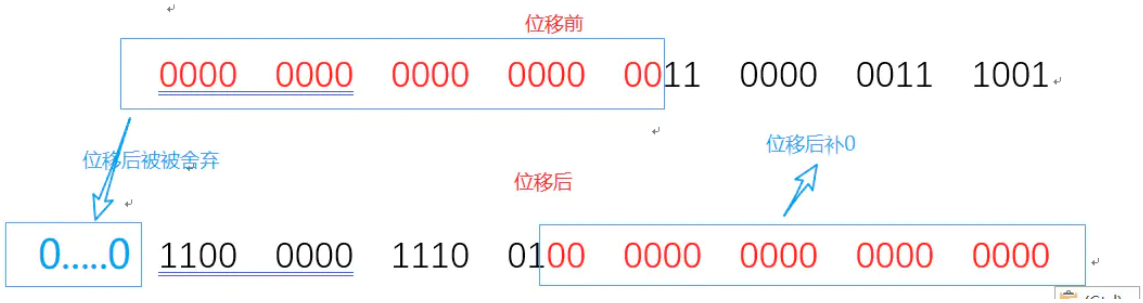
<< 左移运算符

左移一位



左移一位后的数值经过计算可以发现刚好值位移前数值的两倍，等价于乘2操作，在很多情况下可以当做乘2使用，但是并不代表真正的乘2，在一些特殊情况下并不等价

左移18位



此时二进制首位为1，此时数值为 -1058799616，同理，如果左位移20位，则值为 59768832 又变成了正数

> 注意：所以根据这个规则，如果任意一个十进制的数左位移32位，右边补位32个0，十进制岂不是都是0了？当然不是！！！ 当int 类型的数据进行左移的时候，当左移的位数大于等于32位的时候，位数会先求余数，然后用该余数进行左移，也就是说，如果真的左移32位的时候，会先进行位数求余数，即为左移32位相当于左移0位 ，所以左移33 的值和左移一位1 是一样的

>> 右移运算符

100 带符号右移

100 源码补码均为：`00000000 00000000 00000000 01100100`

右移四位：`00000000 00000000 00000000 00000110`

结果为：6

-100 带符号右移

-100原码：`10000000 00000000 00000000 01100100`

-100补码： 保证符号位不变，其余位置取反并加1

`11111111 11111111 11111111 10011100`

右移4位 ： 在高位补1

`11111111 11111111 11111111 11111001`

补码形式的移位完成后，结果不是移位后的结果，还需要进行变换才行。其方法如下:

保留符号位，然后按位取反：`10000000 00000000 00000000 00000110`

然后加1，即为所求数的原码：`10000000 00000000 00000000 00000111`

结果为：-7

>>> 无符号右移运算符

无符号右移运算符和右移运算符是一样的，不过无符号右移运算符在右移的时候是补0的，而右移运算符是补符号位的

100 无符号右移 4 位

100 源码补码均为：`00000000 00000000 00000000 01100100`

右移四位：`00000000 00000000 00000000 00000110`

结果为：6

-100无符号右移4位

-100原码：`10000000 00000000 00000000 01100100`

-100补码：保证符号位不变，其余位置取反并加1

`11111111 11111111 11111111 10011100`

无符号右移4位 ： 在高位补0

`00001111 11111111 11111111 11111001`

结果为：268435449

> 总结：正数的左移与右移、无符号右移、负数的无符号右移，就是相应的补码移位所得，在高位补0即可
>
> 负数的右移，就是补码高位补1,然后按位取反加1即可