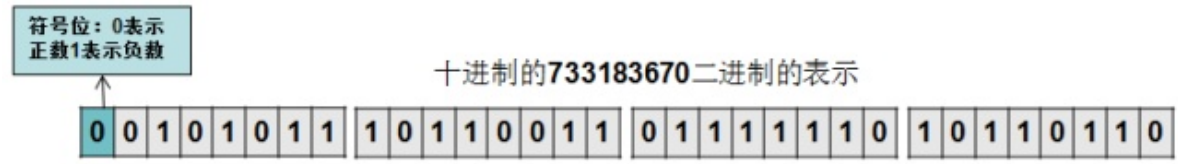
一、位运算

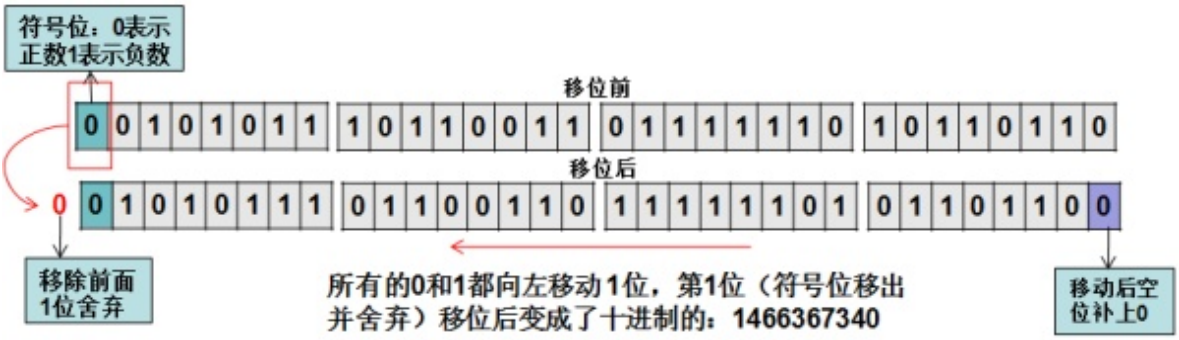
加减乘除取余数

正数：原码、反码、补码相同

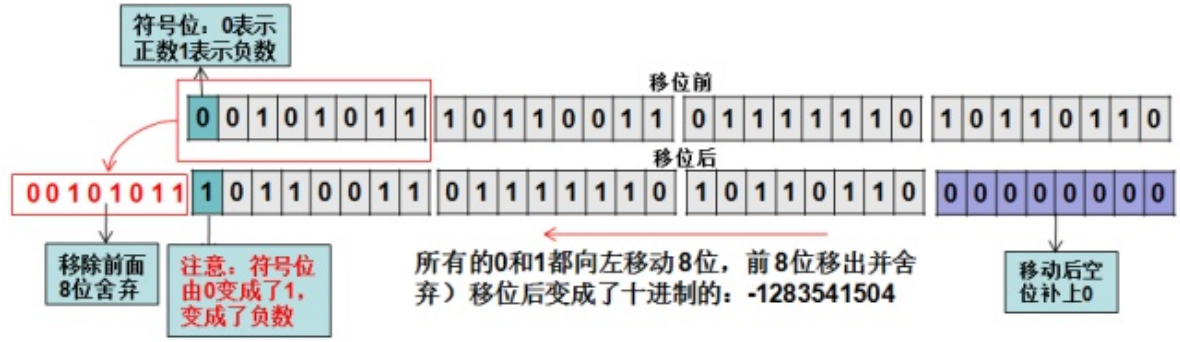
负数：反码是原码除符号位以外的各位求反；补码是原码除符号位以外各位求反之后末位再加1



1、左移运算符 <<



value << 1表示的是变成value的两倍。value = 733183670，进行value<<2运算之后刚好变成了733183670的两倍，有些人在乘2的时候喜欢用左移运算符来替代。



value << 8，左移8位之后，十进制的数变成了-1283541504，移动8位后，由于首位变成了1，也就是说成了负数，在使用中要考虑变成负数的情况。

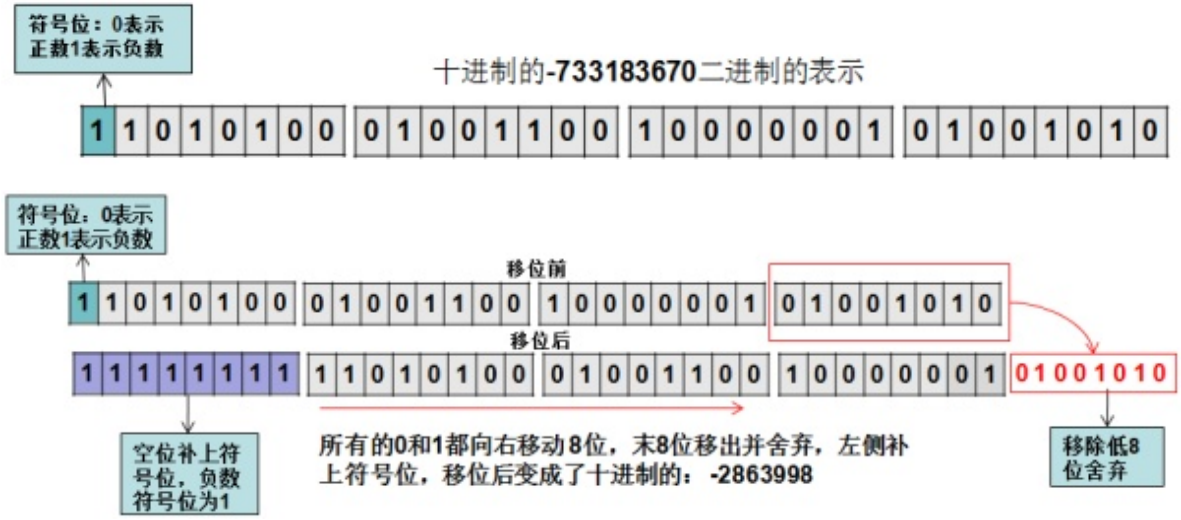
根据这个规则，左移32位后，右边补上32个0值是不是就变成了十进制的0了？**答案是NO，**当int类型进行左移操作时，**左移位数大于等于32位操作时，会先求余（%）后再进行左移操作。**也就是说左移32位相当于不进行移位操作，左移40位相当于左移8位（40%32=8）。当long类型进行左移操作时，long类型在二进制中的体现是64位的，因此求余操作的基数也变成了64，也就是说左移64位相当于没有移位，左移72位相当于左移8位（72%64=8）。

综上所述：左移 << 其实很简单，也就是说**丢弃左边指定位数，右边补0。**

2、右移运算符 >>

右移1位后换算成十进制的值为：366591835，刚好是733183670的1半， 有些人在除2操作时喜欢用右移运算符来替代。

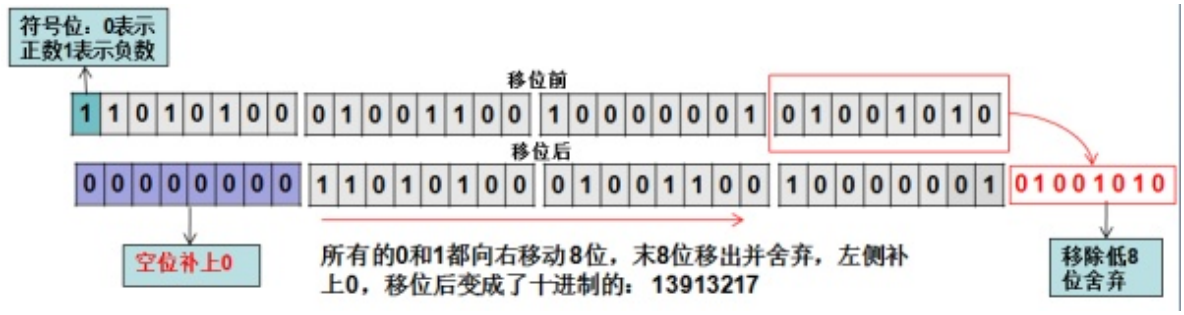
和左移一样，int类型移位大于等于32位时，long类型大于等于64位时，会先做求余处理再位移处理，byte，short移位前会先转换为int类型（32位）再进行移位。以上是正数的位移，我们再来看看负数的右移运算，如图，负数intValue：-733183670的二进制表现如下图：



综上所述：右移运算符>>的运算规则也很简单，丢弃右边指定位数，左边补上符号位。

3、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，只不过右移时左边是补上符号位，而无符号右移运算符是补上0，也就是说，对于正数移位来说等同于：>>，负数通过此移位运算符能移位成正数。以-733183670>>>8为例来画一下图



无符号右移运算符>>的运算规则也很简单，丢弃右边指定位数，左边补上0。

4、无符号左移运算符：<<<

5、按位与&，按位|

&与&&：&表示的是按位与，&&表示的是逻辑与，因此&&不能用在两个数字之间。&&逻辑与也称为短路逻辑与，先运算&&左边的表达式，一旦为假，后续不管多少表达式，均不再计算，一个为真，再计算右边的表达式，两个为真才为真。

> *1 & 10结果为：0*
>
> *false & true结果为：false*
>
> *false && true结果为：false*

|与||：|表示的是按位或，||表示的是逻辑或，因此||不能用在两个数字之间。逻辑或||的运算规则是一个为真即为真，后续不再计算，一个为假再计算右边的表达式。

> *1 | 10结果为：11。1的二进制表示为1,10的二进制表示为1010。*
>
> *false || true结果为：true，先判断第一个为false，如果第一个为false就需要再后续结算。*

6、异或运算符^

^异或运算符顾名思义，异就是不同，其运算规则为1^0 = 1 , 1^1 = 0 , 0^1 = 1 , 0^0 = 0

7、非，取反~

取反就是1为0,0为1,5的二进制位是0000 0101，取反后为1111 1010，值为-6。

二、进制之间的转换

1、十进制转十六进制

十进制转十六进制：Integer.toHexString(int i)

```
    public static String toHexString(int i) {
        return toUnsignedString0(i, 4);
    }

    private static String toUnsignedString0(int val, int shift) {
        // assert shift > 0 && shift <=5 : "Illegal shift value";
        int mag = Integer.SIZE - Integer.numberOfLeadingZeros(val);
        int chars = Math.max(((mag + (shift - 1)) / shift), 1);
        char[] buf = new char[chars];

        formatUnsignedInt(val, shift, buf, 0, chars);

        // Use special constructor which takes over "buf".
        return new String(buf, true);
    }
```

三、格式化输出prinft

https://www.baeldung.com/java-printstream-printf

System.out.printf的重载System.out.printf(format,arguments)和(locate,format,arguments)

System.out.printf(format,arguments)格式化的格式：%[flags][width][.precision]conversion-character

可选项：

- flags：[flags]定义了修改输出的标准方法，并且最常见于格式化整数和浮点数的格式。
- width：[width]指定用于输出参数的字段宽度。 它表示写入输出的最少字符数。
- .precision：[.precision]指定输出浮点值时的精度位数。 另外，我们可以使用它来定义要从字符串中提取的子字符串的长度。

conversion-character的可选值：

- s：表示格式化字符串
- d：表示格式化十进制整数
- f：表示格式化浮点数
- t：表示格式化日期时间值
- ……

换行：%n，%n分隔符printf()将自动插入主机系统的本地行分隔符。

```
System.out.printf("baeldung%nline%nterminator");
结果为：
baeldung
line
terminator
```

Boolean格式化：

- %b，小写输出

```
System.out.printf("%b%n", null);
System.out.printf("%b%n", "random text");
输出结果为：
false
true
```

- %B，大写输出

```
System.out.printf("%B%n", false);
System.out.printf("%B%n", 5.3);
输出结果为：
FALSE
TRUE
```

字符串格式化：%s或%S，设置对齐方式，设置输出字符串长度

```
printf("'%s' %n", "baeldung"); => 'baeldung'
printf("'%S' %n", "baeldung"); => 'BAELDUNG'
printf("'%15s' %n", "baeldung"); => '       baeldung'
// 左对齐，输出长度为10，如果长度超过10照常输出
printf("'%-10s' %n", "baeldung"); => 'baeldung  '
// 我们可以通过指定精度来限制输出中的字符数
// %x.ys中：x代表的是填充，y代表的是输出中的字符数
System.out.printf("%2.2s", "Hi there!"); => Hi
```

字符格式化：%c或%C

```
System.out.printf("%c%n", 's'); => s
System.out.printf("%C%n", 's'); => S
```

整数格式化：用%d来格式化byte、short、int、long和BigInteger，%#o表示八进制表示，%#x或%#X十六进制

```
System.out.printf("simple integer: %d%n", 10000L); => simple integer: 10000
// 如果我们需要对数字按照千分割，可以使用可选项[flag]可以用','来表示，同时也可以使用不同的locales来获得本地化表示！！！注意必须有[flag]才可以！！！
System.out.printf("%,d %n", 10000); => 10,000
System.out.printf(Locale.US, "%,d %n", 10000); => 10,000
System.out.printf(Locale.ITALY, "%,d %n", 10000); => 10.000
```

float和double类型格式化：%f或%e或%E

```
System.out.printf("%f%n", 5.1473); => 5.147300
// %x.yf，x表示的是宽度，y表示的是小数部分的长度（x宽度包括y的长度），四舍五入的结果
System.out.printf("'%5.2f'%n", 5.1473); => ' 5.15'
System.out.printf("'%5.2e'%n", 5.1473); => '5.15e+00'
System.out.printf("'%5.2E'%n", 5.1473); => '5.15E+00'
```

Time格式化：在java8中提供了DateTimeFormatter来完成格式化

- H，M，S分别代表从输入Date中提取小时、分钟、秒
- L，N：代表获取milliseconds and nanoseconds
- p：添加am/pm格式化
- z：表述输出时区偏移

```
Date date = new Date();
System.out.printf("%tT%n", date); => 13:51:15
System.out.printf("hours %tH: minutes %tM: seconds %tS%n", date, date, date); => hours 13: minutes 51: seconds 15
System.out.printf("%1$tH:%1$tM:%1$tS %1$tp %1$tL %1$tN %1$tz %n", date); => 13:51:15 pm 061 061000000 +0400
```

Date格式化：

- A：输出当前日期在week中的星期全称
- d：格式化一个两位数的月中第几天
- B：输出月份英文全称

- m：输出两位数月份
- Y：输出四位数年份
- y：输出年份后两位数

```
System.out.printf("%1$tA, %1$tB %1$tY %n", date); => Thursday, November 2018
System.out.printf("%1$td.%1$tm.%1$ty %n", date); => 22.11.18
```

四、Arrays类

1、排序相关

2、拷贝相关

3、

五、泛型

E - Element (在集合中使用，因为集合中存放的是元素)

T - Type（Java 类）

K - Key（键）

V - Value（值）

N - Number（数值类型）

？ - 表示不确定的java类型

S、U、V - 2nd、3rd、4th types

Object跟这些标记符代表的java类型有啥区别呢？

Object是所有类的根类，任何类的对象都可以设置给该Object引用变量，使用的时候可能需要类型强制转换，但是用使用了泛型T、E等这些标识符后，在实际用之前类型就已经确定了，不需要再进行类型强制转换。

六、类加载机制