

什么是位运算，你了解 JavaScript 中的位运算么？

在了解位运算之前，我们先必须了解一下什么是原码、反码和补码以及二进制与十进制的转换。

原码、补码和反码

原码

一个数在计算机中是以二进制的形式存在的，其中第一位存放符号，正数为 0，负数为 1。原码就是用第一位存放符号的二进制数值。例如 2 的原码为 00000010，-2 的原码为 10000010。

反码

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，其余各位取反，即 0 变 1，1 变 0。

$[+3] = [00000011]$ 原 = $[00000011]$ 反

$[-3] = [10000011]$ 原 = $[11111100]$ 反

可见如果一个反码表示的是负数，并不能直观的看出它的数值，通常要将其转换成原码再计算。

补码

正数的补码是其本身。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。（即负数的补码为在其反码的基础上 +1）。

$[+3] = [00000011]$ 原 = $[00000011]$ 反 = $[00000011]$ 补

$[-3] = [10000011]$ 原 = $[11111100]$ 反 = $[11111101]$ 补

可见对于负数，补码的表示方式也是让人无法直观看出其数值的，通常也需要转换成原码再计算。

二进制与十进制的转换

二进制与十进制的区别在于数运算时是逢几进一位。二进制是逢 2 进一位，十进制也就是我们常用的 0-9 是逢 10 进一位

正整数的十进制转二进制

正整数的十进制转二进制的方法为将一个十进制数除以 2，得到的商再除以 2，以此类推直到商等于 1 或 0 时为止，倒取除得的余数，即为转换所得的二进制数的结果。

例如把 52 换算成二进制数，计算过程如下图：

```

2 | 52 .....0
  | 26 .....0
  | 13 .....1
  |  6 .....0
  |  3 .....1
  |  1 .....1
  
```

52 除以 2 得到的余数依次为：0、0、1、0、1、1，倒序排列，所以 52 对应的二进制数就是 110100。

负整数的十进制转二进制

负整数的十进制转二进制为将该负整数对应的正整数先转换成二进制，然后对其“取反”，再对取反后的结果+1。即负整数采用其二进制补码的形式存储。

至于负数为什么要用二进制补码的形式存储，可参考一篇阮一峰的文章《关于 2 的补码》。例如 -52 的原码为 10110100，其反码为 11001011，其补码为 11001100。所以 -52 转换为二进制后为 11001100。

十进制小数转二进制

十进制小数转二进制的方法为“乘 2 取整”，对十进制小数乘 2 得到的整数部分和小数部分，整数部分即是相应的二进制数码，再用 2 乘小数部分(之前乘后得到新的小数部分)，又得到整数和小数部分。如此不断重复，直到小数部分为 0 或达到精度要求为止。第一次所得为最高位，最后一次得到为最低位。

如:0.25 的二进制

$0.25 \times 2 = 0.5$ 取整是 0

$0.5 \times 2 = 1.0$ 取整是 1

即 0.25 的二进制为 0.01 (第一次所得为最高位,最后一次得到为最低位)

0.8125 的二进制

$0.8125 \times 2 = 1.625$ 取整是 1

$0.625 \times 2 = 1.25$ 取整是 1

$0.25 \times 2 = 0.5$ 取整是 0

$0.5 \times 2 = 1.0$ 取整是 1

即 0.8125 的二进制是 0.1101

二进制转十进制

从最后一位开始算，依次列为第 0、1、2... 位，第 n 位的数 (0 或 1) 乘以 2 的 n 次方，将得到的结果相加就是得到的十进制数。

例如二进制为 110 的数，将其转为十进制的过程如下

$$\begin{array}{ccc} 1 & 1 & 0 \\ \hline 2^2 & 2^1 & 2^0 \end{array}$$

个位数 0 与 2^0 相乘： $0 \times 2^0 = 0$

十位数 1 与 2^1 相乘： $1 \times 2^1 = 2$

百位数 1 与 2^2 相乘： $1 \times 2^2 = 4$

将得到的结果相加： $0+2+4=6$

所以二进制 110 转换为十进制后的数值为 6。

小数二进制用数值乘以 2 的负幂次然后相加。

同样，取反也可以做向下取整运算，因为当 x 为整数时有 $\sim x == x$ ，所以当 x 为小数时有 $\sim x == \text{Math.floor}(x)$ 。

```
function isPos(n) {
    return (n === (n >>> 0)) ? true : false;
}
```

```
isPos(-1); // false
```

```
isPos(1); // true
```

总结

根据 JS 的位运算，可以得出如下信息：

- 1、所有的位运算都可以对小数取底。
- 2、对于按位与 $\&$ ，可以用 $(n \& 1) === 0$ //true 为偶数，false 为奇数。来判断奇偶。用 $x \& -1 === \text{Math.floor}(x)$ 来向下取底。
- 3、对于按位或 $|$ ，可以用 $x|0 === \text{Math.floor}(x)$ 来向下取底。
- 4、对于取反运算 \sim ，可以用 $\sim\sim x === \text{Math.floor}(x)$ 来向下取底。
- 4、对于异或运算 \wedge ，可以用 $(x^0) === \text{Math.floor}(x)$ 来向下取底。
- 4、对于左移运算 \ll ，可以 $x \ll n === x * 2^n$ 来求 2 的 n 次方，用 $x \ll 0 === \text{Math.floor}(x)$ 来向下取底。
- 4、对于有符号右移运算 \gg ，可以 $x \gg n === x / 2^n$ 求一个数字的 N 等分，用 $x \gg 0 === \text{Math.floor}(x)$ 来向下取底。
- 4、对于无符号右移运算 \ggg ，可以 $(n === (n \ggg 0)) ? \text{true} : \text{false}$; 来判断数字正负，用 $x \ggg 0 === \text{Math.floor}(x)$ 来向下取底。

用移位运算来替代普通算术能获得更高的效率。移位运算翻译成机器码的长度更短，执行更快，需要的硬件开销更小。

