

# 二进制运算符

二进制运算符做为选讲内容

二进制运算符也叫位运算符，它按数值的位来进行运算，在 **ECMAScript** 的系统里面，**所有的数据都是用64位存储，32位运算**，如果进行二进制的位运算，需要先将原来的数值转换成2进制操作

位操作符用于在最基本的层次上，即按内存中表示数值的位来操作数值。**ECMAScript 中的所有数值都以 IEEE-754 64 位格式存储，但位操作符并不直接操作 64 位的值。而是先将 64 位的值转换成 32 位的整数，然后执行操作，最后再将结果转换回 64 位。**对于开发人员来说，由于 64 位存储格式是透明的，因此整个过程就像是只存在 32 位的整数一样。

## 按位非（NOT）

按位非操作符由一个波浪线（~）表示，执行按位非的结果就是返回数值的反码。按位非是 ECMAScript 操作符中少数几个与二进制计算有关的操作符之一。

```
1 var num1 = 25;
2 var num2 = ~num1;
```

现在来看它的计算过程

### 第一步：先将这个数转2进制

```
1 num1.toString(2);    //11001
```

### 第二步：补码

位数	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
25的二进制的补码	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1

### 第三步：取反

位数	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
25的二进制的补码	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1
反码	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0

取反以后，我们发现它的最高位31位是一个数字 1 ,这就说明这个数是一个负数，**负数的二进制是需要进行转码的**

### 第三步：将负数的二进制转码

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG
位数	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
25的二进制	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1
反码	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0
减1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0	1
保留符号再取反	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0

将负数的二进制转码以后，得到的结果是 11010 ,但是它的符号位是 1 说明它是一个负数

```
1 parseInt("11010",2);    //26
2 //因为符号位是负数，所以最终的结果是-26
```

通过上面计算，其实我们发现二进制的取反过程它就是将原来的数乘以 -1 再减去1就可以

# 按位与（AND）

按位与操作符使用的符号是 `&`

```
1  var result = 18 & 3;
```

这个操作过程也很简单

位数	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
18的二进制	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0
3的二进制	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
按位与	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

第一个数值的位	第二个数值的位	结 果
1	1	1
1	0	0
0	1	0
0	0	0

# 按位或（OR）

按位或操作符由一个竖线符号 (`|`) 表示，同样也有两个操作数。按位或操作遵循下面这个真值表

第一个数值的位	第二个数值的位	结 果
1	1	1
1	0	1
0	1	1
0	0	0

```
1  var result = 23 | 71;
```

位数	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
23的二进制	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1
71的二进制	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	1
按位或	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	1	1

得到结果87

## 小练习

```
1  var result1 = ~~17;           //16
2
3  var result2 = -16 & 8;         //0
4
5  var result3 = -22 & -18;       //-22
6
7  var result4 = -4 | 5;
8
9  var result5 = -5 | -6;
```

上面的练习里面有负数，对于负数的二进制要以补码的形式去完成

图灵社区会员 StinkBC(StinkBC@gmail.com) 专享 尊重版权

- (1) 求这个数值绝对值的二进制码 (例如, 要求-18 的二进制补码, 先求 18 的二进制码);
  - (2) 求二进制反码, 即将 0 替换为 1, 将 1 替换为 0;
  - (3) 得到的二进制反码加 1。
- 要根据这 3 个步骤求得-18 的二进制码, 首先就要求得 18 的二进制码, 即:

## 按位异或 (XOR)

按位异或操作符由一个插入符号 (^) 表示，也有两个操作数。以下是按位异或的真值表。

第一个数值的位	第二个数值的位	结 果
1	1	0
1	0	1
0	1	1
0	0	0

```
1 var result = 25 ^ 3; //26
```

## 左移

按位左移运算符使用 `<<` 来表示

```
1 var a = 2;  
2 var b = a<<5; //64
```

位数	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
2的2进制	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
左移5位	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	

## 有符号的右移

有符号的右移操作符由两个大于号 (>>) 表示，这个操作符会将数值向右移动，**但保留符号位**（即 正负号标记）。有符号的右移操作与左移操作恰好相反。

```
1 var a = 64;  
2 var b = a>>5;
```

[illegible]

# 无符号右移

无符号右移操作符由 3 个大于号 (>>>) 表示，这个操作符会将数值的所有 32 位都向右移动。对正数来说，无符号右移的结果与有符号右移相同，但对于负数来说，这个结果是会有影响的，因为负数的最高位是 1

```
1  var a = -64;
2  var b = a>>>5;           //134,217,726
```

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH
位数		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
负64的2进制码		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	
无符号右移		0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	

## 注意事项

- 1. 所有二进制的操作符都不对小数进行操作，后期我们可以复用这个特点来快速的将一个数值取整

```
1  var a = 3.14;
2  var b = ~~a;           //3;
```

- 2. 左移与右移运算符不得超过原有的计算机位数，如果超过了，将返回一个毫无意义的数