算数运算符

1. 加法运算符

一个表达式中，如果至少有一个运算子是字符串，使用加法运算符后，表达式最终也返回字符串。

1.1. 求两数之和

如果两个运算子都不是字符串的原始类型，则返回数值。

```
1 + 1;       // 2
true + true; // 2
1 + true;    // 2
```

1.2. 连接运算符

两个字符串相加，这时加法运算符会变成连接运算符，返回一个新的字符串，将两个原字符串连接在一起。

```
"a" + "b"; // "ab"
```

如果一个运算子是字符串，另一个运算子是非字符串，这时非字符串会转成字符串，再连接在一起。

```
1 + "a";     // "1a"
3 + 4 + "5"; // "75"
"3" + 4 + 5; // "345"
```

也就是说，运算子的不同，导致了不同的语法行为，这种现象称为“重载”（overload）。

2. 减乘除运算符

除了加法运算符会发生重载，其他运算符（减法、乘法、除法）都不会发生重载。所有运算子一律转换为数值，再进行相应的数学运算。

```
2 - 1;     // 1
"2" - "1"; // 1
"a" - "b"; // NaN
// <=> Number("a") - Number("b")
// <=> NaN - NaN
```

`"a"` 和 `"b"` 无法转换成数值，所以是 `NaN`。

```
1 * 5;            // 5
true * false;     // 0 (<=>  1 * 0 )
"true" * "false"; // NaN (<=>  NaN * NaN )
```

"true" 和 "false" 无法转换成数值，所以是 NaN。

```
5 / 5;        // 1
"40" / "8";   // 5
true / false; // Infinity (<=>  1 / 0)
false / true; // 0 (<=> 0 / 1)
```

3. 余数运算符

运算结果的正负号由第一个运算子的正负号决定。

```
-1 % 2; // -1
1 % -2; // 1
```

4. 自增自减运算符

自增或自减会使原来的变量发生改变，这种效应是运算的副作用。**自增和自减运算符是仅有的两个具有副作用的运算符，其他运算符都不会改变变量的值。**

```
let x = 1;
++x; // 2
x;   // 2

--x; // 1
x;   // 1
```

5. 数值运算符，负数值运算符

数值运算符 + 只需要一个操作数，需要两个操作数的是加法运算符。**数值运算符的作用是将任何值转换为数值（与 Number 函数的作用相同）。**

```
+0;         // 0
+1;         // 1
+"a";       // NaN
+"1";       // 1
+true;      // 1
+false;     // 0
+NaN;       // NaN
+undefined; // NaN
```

```
+null;      // 0
+{};        // NaN
+[];        // 0
+Infinity;  // Infinity
```

`Infinity` 和 `NaN` 是特殊的数值：

```
typeof Infinity; // "number"
typeof NaN;      // "number"
```

数值运算符号和负数值运算符，都会返回一个新的值，而不会改变原始变量的值。

```
let x = "3";
+x; // 3
x;  // "3"
```

6. 指数运算符

指数运算符（**）完成指数运算，前一个运算子是底数，后一个运算子是指数。

```
2 ** 4; // 16 (<=> Math.pow(2, 4))
```

指数运算符是右结合，而不是左结合。即多个指数运算符连用时，先进行最右边的计算。

```
(2 ** 3) ** 3; // 512
2 ** (3 ** 3); // 134217728  <=> 2 ** 3 ** 3;
```

7. 赋值运算符

赋值运算符可以和其他运算符结合。

```
x += y;  // <=> x = x + y;
x -= y;  // <=> x = x - y;
x *= y;  // <=> x = x * y;
x /= y;  // <=> x = x / y;
x %= y;  // <=> x = x % y;
x **= y; // <=> x = x ** y;
x ||= y; // <=> x || (x = y)
x &&= y; // <=> x && (x = y);
x ??= y; // <=> x ?? (x = y);
```