语言基础

操作符

ES中操作符是独特的，包括字符串、数值、布尔值，甚至还有对象。在应用给对象时，操作符通常会调用valueOf()和/或toString()方法来取得可以计算的值。

一元操作符

1.递增/递减操作符

```
let i = 0
let b = i++
console.log(b) // 0

b = ++i
console.log(b) // 2
```

i，先操作，后访问
i，先访问，后操作

- 字符串，有效数值形式，转数值在操作
- 字符串，无效数值形式，NaN
- Boolean， false->0,true->1
- 对象，调用valueOf,可以用上述规则就用，如果是NaN就toString()再应用规则

```
let s1 = "2";
let s2 = "z";
let b = false;
let f = 1.1;
let o = {
 valueOf() {
 return -1;
 }
};
s1++; // 值变成数值 3
s2++; // 值变成 NaN
b++; // 值变成数值 1
f--; // 值变成 0.100
```

2.一元加和减

- 正负
- 非数值则Number()

```
let s1 = "01";
let s2 = "1.1";
let s3 = "z";
let b = false;
let f = 1.1;
let o = {
 valueOf() {
 return -1;
 }
};
s1 = +s1; // 值变成数值 1
s2 = +s2; // 值变成数值 1.1
s3 = +s3; // 值变成 NaN
b = +b; // 值变成数值 0
f = +f; // 不变，还是 1.1
o = +o; // 值变成数值-1,默认调用valueOf函数，但是被改写了
```

位操作符

负值以二补码表示

1. 确定绝对值的二进制表示
2. 反码
3. +1

- 按位非 ~
  32位全反
- 按位与 &
  两个位都是1返回1
- 按位或 |
  有一个位是1返回1
- 按位异或 ^
  两个位不同返回1
- 左移 <<
- 有符号右移 >>
- 无符号右移 >>>

布尔操作符

- 逻辑非 !
  - 对象， false
  - 空字符串，true
  - 非空字符串，false
  - 0，true
  - 非零，false
  - null，true
  - NaN，true
  - undefined，true
- 逻辑与 &&
  如果有一个操作数不是布尔值!

    i. 第一个操作数是对象，返回第二个操作数
    ii. 第二个操作数是对象，只有第一个操作数位true才会返回该对象,否则返回第一个
    iii. 有一个是null，返回null
    iv. 有一个NaN，返回NaN
    v. 有一个undefined，返回undefined
    vi. 两个都属于null/NaN/undefined，返回第一个
- 逻辑或 ||
  如果有一个操作数不是布尔值!
    i. 第一个操作符是对象，则返回第一个
    ii. 第一个false，返回第二个
    iii. 两个都是null/NaN/undefined，返回第二个（即第二条）

乘性操作符

- 乘法操作符 *
    i. 任何数操作NaN，返回NaN
    ii. Infinity*0，返回NaN
    iii. 如果不是数值，Number()
- 除法操作符 /
    i. 如果是 Infinity 除以 Infinity，则返回 NaN
    ii. 0/0,NaN
    iii. 非零有限值/0，Infinity或-Infinity
    iv. 如果不是数值，Number()
- 取模操作符 %
    i. 被除数无限制，除数有限值，NaN
    ii. 被除数有限值，除数0，NaN
    iii. Infinity%Infinity，NaN
    iv. 被除数是有限值，除数是无限值，则返回被除数
    v. 如果被除数是 0，除数不是 0，则返回 0
    vi. 不是数值，Number()

指数操作符

ES7新增了指数操作符，Math.pow()，可以用**

```
console.log(Math.pow(3, 2); // 9
console.log(3 ** 2); // 9
console.log(Math.pow(16, 0.5); // 4
console.log(16** 0.5); // 4

let squared = 3;
squared **= 2;
console.log(squared); // 9
```

加性操作符

- 加法操作符
    i. Infinity + -Infinity，NaN

ii. +0 + +0，+0
iii. -0 + +0，+0
iv. -0 + -0，-0
v.
    - null/udefined，NaN

- 减法操作符
    i. Infinity - Infinity,NaN
    ii. -Infinity - -Infinity,NaN
    iii. +0 - +0,+0
    iv. +0 - -0,-0
    v. -0 - -0,+0

第四条太反人类了

关系操作符

< > <= >=
返回布尔值

- 都是数值，数值比较
- 都是字符串，逐个比较字符串中对应字符的编码
- 有一个数数值，另一个转换为数值，按数值比较
  任何关系操作符涉及NaN,都返回false
- 任一操作数是对象，调用valueOf(),如果没有valueOf()调用toString()
- 任一操作数是布尔值，将其转为数值再去比较

相等操作符

- 等于和不等于 == !=
    i. boolean，转数值后在比较
    ii. 字符串，数值---->字符串转数值
    iii. object，数值---->先valueOf(),然后转数值比较
    iv. null == undefined,不转数值
    v. NaN, 永远不等
    vi. 都是对象，是否指向同一个地址

```
null == undefined                    true
"NaN" == NaN                         false
5 == NaN                             false
NaN == NaN                           false
NaN != NaN                           true
false == 0                           true
true == 1                            true
true == 2                            false
undefined == 0                       false
null == 0                            false
"5" == 5                             true
```

- 全等和全不等 === !==
  比较时不转换操作数

条件操作符

```
let max = (num1 > num2) ? num1 : num2;
```

赋值操作符

用等于，把右边的值给左边的变量

```
let num = 10;
```

复合赋值操作

- 乘后赋值 *=
- 除后赋值 /=
- 取模后赋值 %=
- 加后赋值 +=
- 减后赋值 -=
- 左移后赋值 <<=
- 右移后赋值 >>=
- 无符号右移后赋值 >>>=

这些操作符仅仅是简写语法，使用它们不会提升性能。

逗号操作符

```
let num1 = 1, num2 = 2, num3 = 3;
let num = (5, 1, 4, 8, 0); // num 的值为 0
```

第二条很反人类