比较运算符

比较运算符分为两类：

> - 相等比较
> - 非相等比较

相等比较分为严格和不严格，非相等运算符分为字符串的比较和非字符串的比较。

1. 非相等比较

1.1. 字符串的比较

字符串按照字典顺序进行比较。

```
"c" > "d"; // false
"c" > "b"; // true
```

首先比较首字母，如果相等，再比较第二个字母，以此类推。

```
"cat" > "cbd"; // false
"dom" > "dog"; // true
```

如果是同一个字母，小写的字母 `Unicode` 码点大于大写的 `Unicode` 的码点。

```
"c" > "C"; // true
"Man" > "man"; // false
```

因为所有的字符都有 Unicode 码，所以汉字也可以比较。

```
"英" > "汉"; //  true
```

1.2. 非字符串的比较

如果两个运算子中，不全是字符串（即至少有一个不是字符串），需要分情况。

1.2.1. 原始类型值

如果两个运算子都是原始类型值，则先转换成数值再比较。

```
5 > "6"; // false
// 等价于 5 > Number("6"), 5 > 6

0 > true; // false
// 等价于 0 > Number(true), 0 > 1

"1" >= true; // true
// 等价于 Number("1") >= Number(true), 1 >= 1

true > false; // true
// 等价于 Number(true) > Number(false), 1 > 0
```

任何值（包括 NaN 本身）与 NaN 使用非相等运算符进行比较，返回的都是 false。

```
1 > NaN; // false
1 <= NaN; // false
"1" > NaN; // false
"1" <= NaN; // false
true > NaN; // false
true <= NaN; // false
false > NaN; // false
false <= NaN; // false
NaN > NaN; // false
NaN <= NaN; // false
```

1.2.2. 对象

如果运算子是对象，会转为原始类型的值，再进行比较。

对象转换成原始类型的值，算法是先调用 valueOf 方法；如果返回的还是对象，再接着调用 toString 方法。

```
[2] > [1]; // true
// 等价于 [2].valueOf().toString() > [1].valueOf().toString(), '2' > '1'

{a: 1} >= {a: 2};  // true
// 等价于 ({a: 1}).valueOf().toString() > ({a: 2}).valueOf().toString(), "[object
Object]" >= "[object Object]"

let f1 = function() { return 'a' };
let f2 = function() { return 'b' };
f1 < f2;  // true
// 等价于 f1.valueOf().toString() < f2.valueOf().toString(), "function(){return
'a'}" < "function(){return 'b'}"

let a = [2];
a.valueOf = function() { return '1' };
a > '11' // false
// 等价于 a.valueOf() > '11', '1' > '11'
```

2. 相等比较

2.1. 严格相等

2.1.1. 元素类型

`==` 和 `===` 的区别是，前者比较两个值是否相等，后者比较他们是否为“同一个值”。如果两个值不是同一种类型，前者会将其转换成同一种类型再比较，后者直接返回 `false`。

`NaN` 与任何值都不相等，包括自身，`+0` 等于 `-0` 。

```
NaN === NaN; // false
+0 === -0; // true
```

类型相同，值相等就认为严格相等。

```
1 === 0x1; // true  类型都是数值类型，值都是 1（后面是16进制的 1）
```

2.1.2. 复合类型

两个复合类型（对象、数组、函数）的数据比较时，不是比较它们的值是否相等，而是比较它们是否指向同一个地址。

```
{} === {} // false
[] === [] // false
(function () {} === function () {}) // false
```

运算符两边的空对象、空数组、空函数的值，都存放在不同的内存地址，结果当然是 `false`。

如果两个变量引用同一个对象，则它们相等。

```
let v1 = {};
let v2 = v1;
v1 === v2; // true
```

2.1.3. null 和 undefined

`undefined` 和 `null` 与自身严格相等。

```
undefined === undefined; // true
null === null; // true
```

由于变量声明后默认值是 `undefined`，因此两个只声明未赋值的变量是相等的。

```
let v1;
let v2;
v1 === v2; // true
```

2.2. 严格不相等

严格相等运算符有一个对应的“严格不相等运算符”（!==），它的算法就是先求严格相等运算符的结果，然后返回相反值。

```
1 !== "1"; // true
// 等价于
!(1 === "1"); // true
```