布尔运算符

1. 取反运算符（!）

取反运算符是一个感叹号，用于将布尔值变为相反值，即 `true` 变成 `false` ， `false` 变成 `true`。

```
!undefined // true
!null // true
!0 // true
!NaN // true
!"" // true
!false // true

![] // false
!{} // false
```

如果对一个值连续做两次取反运算，等于将其转为对应的布尔值，与 `Boolean` 函数的作用相同。 两次取反就是将一个值转为布尔值的简便写法。

```
!!x
// <=>
Boolean(x)
```

2. 且运算符（&&）

且运算符（&&）往往用于多个表达式的求值。 **如果第一个运算子的布尔值为 `true`，则返回第二个运算子的值（不是布尔值）；如果第一个运算子的布尔值为 `false`，则直接返回第一个运算子的值，且不再对第二个运算子求值。**

```
't' && '' // ""
't' && 'f' // "f"
't' && (1 + 2) // 3
'' && 'f' // ""
'' && '' // ""

let x = 1;
(1 - 1) && ( x += 1) // 0
x // 1，第一个运算子的布尔值为 false，则直接返回它的值 0，而不再对第二个运算子求值，所以变量 x 的值没变。
```

这种跳过第二个运算子的机制，被称为“短路”。可用它取代 `if` 结构：

```

```

```
if (i) {
  doSomething();
}
// <=>
i && doSomething();
```

上面代码的两种写法是等价的，但是后一种不容易看出目的，也不容易排查。

且运算符可以多个连用，这时返回第一个布尔值为 `false` 的表达式的值。如果所有表达式的布尔值都为 `true`，则返回最后一个表达式的值。

```
true && 'foo' && '' && 4 && 'foo' && true
// '', 第一个布尔值为 false 的表达式为第三个表达式，所以得到一个空字符串。

1 && 2 && 3
// 3，所有表达式的布尔值都是 true，所以返回最后一个表达式的值 3。
```

3. 或运算符（||）

如果第一个运算子的布尔值为 `true`，则返回第一个运算子的值，且不再对第二个运算子求值；如果第一个运算子的布尔值为 `false`，则返回第二个运算子的值。

```
't' || '' // "t"
't' || 'f' // "t"
'' || 'f' // "f"
'' || '' // ""
```

短路规则对这个运算符也适用。

```
let x = 1;
true || (x = 2) // true
x // 1，第一个运算子为 true，所以直接返回 true，不再运行第二个运算子，x 的值没有改变
```

只通过第一个表达式的值，控制是否运行第二个表达式的机制，就称为“短路”（short-cut）。

或运算符可以多个连用，这时返回第一个布尔值为 `true` 的表达式的值。如果所有表达式都为 `false`，则返回最后一个表达式的值。

```
false || 0 || '' || 4 || 'foo' || true
// 4，第一个布尔值为 true 的表达式是第四个表达式，所以得到数值 4。

false || 0 || ''
// ''，所有表达式的布尔值都为 false，所以返回最后一个表达式的值。
```

3. 三元条件运算符（?:）

三元条件运算符由问号（?）和冒号（:）组成，分隔三个表达式。它是 JavaScript 语言唯一一个需要三个运算子的运算符。 **如果第一个表达式的布尔值为 `true`，则返回第二个表达式的值，否则返回第三个表达式的值。**

```
't' ? 'hello' : 'world' // "hello"
0 ? 'hello' : 'world' // "world"
```

三元条件表达式与 `if...else` 语句具有同样表达效果，前者可以表达的，后者也能表达。但是两者具有一个重大差别，`if...else` **是语句，没有返回值；三元条件表达式是表达式，具有返回值。** 所以，在需要返回值的场合，只能使用三元条件表达式，而不能使用 `if..else`。

```
console.log(true ? 'T' : 'F'); // T，console.log 需要返回值，这里使用三元条件表达式
```