其他运算符，运算顺序

目录
void 运算符
逗号运算符
运算顺序
优先级
圆括号的作用
左结合与右结合

void 运算符
void 运算符的作用是执行一个表达式，然后不返回任何值，或者说返回 undefined。

void 0 // undefined
void(0) // undefined

上面是 void运算符的两种写法，都正确。建议采用后一种形式，即总是使用圆括号。因为 void
运算符的优先性很高，如果不使用括号，容易造成错误的结果。比如， void 4 + 7实际上等同于 (void 4) + 7 。

下面是 void运算符的一个例子。

var x = 3;
void (x = 5) //undefined
x // 5

这个运算符的主要用途是浏览器的书签工具（ Bookmarklet），以及在超级链接中插入代码防止网页跳转。

请看下面的代码。

<script>
function f() {
console.log('Hello World');

```
}
</script>
<a href="http://example.com" onclick="f(); return false;">点击</a>
```

上面代码中，点击链接后，会先执行 `onclick` 的代码，由于 `onclick` 返回 `false`，所以浏览器不会跳转到 example.com。

`void` 运算符可以取代上面的写法。

```
<a href="javascript: void(f())">文字</a>
```

下面是一个更实际的例子，用户点击链接提交表单，但是不产生页面跳转。

```
<a href="javascript: void(document.form.submit())">
  提交
</a>
```

逗号运算符

逗号运算符用于对两个表达式求值，并返回后一个表达式的值。

```
'a', 'b' // "b"

var x = 0;
var y = (x++, 10);
x // 1
y // 10
```

上面代码中，逗号运算符返回后一个表达式的值。

逗号运算符的一个用途是，在返回一个值之前，进行一些辅助操作。

```
var value = (console.log('Hi!'), true);
// Hi!

value // true
```

上面代码中，先执行逗号之前的操作，然后返回逗号后面的值。

运算顺序

优先级

JavaScript 各种运算符的优先级别（Operator Precedence）是不一样的。优先级高的运算符先执行，优先级低的运算符后执行。

```
4 + 5 * 6 // 34
```

上面的代码中，乘法运算符（`*`）的优先性高于加法运算符（`+`），所以先执行乘法，再执行加法，相当于下面这样。

```
4 + (5 * 6) // 34
```

如果多个运算符混写在一起，常常会导致令人困惑的代码。

```
var x = 1;
var arr = [];

var y = arr.length <= 0 || arr[0] === undefined ? x : arr[0];
```

上面代码中，变量`y`的值就很难看出来，因为这个表达式涉及5个运算符，到底谁的优先级最高，实在不容易记住。

根据语言规格，这五个运算符的优先级从高到低依次为：小于等于（`<=`）、严格相等（`===`）、或（`||`）、三元（`?:`）、等号（`=`）。因此上面的表达式，实际的运算顺序如下。

```
var y = ((arr.length <= 0) || (arr[0] === undefined)) ? x : arr[0];
```

记住所有运算符的优先级，是非常难的，也是没有必要的。

圆括号的作用

圆括号（`()`）可以用来提高运算的优先级，因为它的优先级是最高的，即圆括号中的表达式会第一个运算。

```
(4 + 5) * 6 // 54
```

上面代码中，由于使用了圆括号，加法会先于乘法执行。

运算符的优先级别十分繁杂，且都是硬性规定，因此建议总是使用圆括号，保证运算顺序清晰可读，这对代码的维护和除错至关重要。

顺便说一下，圆括号不是运算符，而是一种语法结构。它一共有两种用法：一种是把表达式放在圆括号之中，
提升运算的优先级；另一种是跟在函数的后面，作用是调用函数。

注意，因为圆括号不是运算符，所以不具有求值作用，只改变运算的优先级。

var x = 1;
(x) = 2;

上面代码的第二行，如果圆括号具有求值作用，那么就会变成 1 = 2 ，这是会报错了。但是，上面的代码可以运行，这验证了圆括号只改变优先级，不会求值。

这也意味着，如果整个表达式都放在圆括号之中，那么不会有任何效果。

(expression)
// 等同于
expression

函数放在圆括号中，会返回函数本身。如果圆括号紧跟在函数的后面，就表示调用函数。

function f() {
return 1;
}

(f) // function f(){return 1;}
f() // 1

上面代码中，函数放在圆括号之中会返回函数本身，圆括号跟在函数后面则是调用函数。

圆括号之中，只能放置表达式，如果将语句放在圆括号之中，就会报错。

(var a = 1)
// SyntaxError: Unexpected token var

左结合与右结合

对于优先级别相同的运算符，同时出现的时候，就会有计算顺序的问题。

a OP b OP c

上面代码中，`OP`表示运算符。它可以有两种解释方式。

```
// 方式一
(a OP b) OP c

// 方式二
a OP (b OP c)
```

上面的两种方式，得到的计算结果往往是不一样的。方式一是将左侧两个运算数结合在一起，采用这种解释方式的运算符，称为“左结合”（left-to-right associativity）运算符；方式二是将右侧两个运算数结合在一起，这样的运算符称为“右结合”运算符（right-to-left associativity）。

JavaScript 语言的大多数运算符是“左结合”，请看下面加法运算符的例子。

```
x + y + z

// 引擎解释如下
(x + y) + z
```

上面代码中，`x`与`y`结合在一起，它们的预算结果再与`z`进行运算。

少数运算符是“右结合”，其中最主要的是赋值运算符（`=`）和三元条件运算符（`?:`）。

```
w = x = y = z;
q = a ? b : c ? d : e ? f : g;
```

上面代码的解释方式如下。

```
w = (x = (y = z));
q = a ? b : (c ? d : (e ? f : g));
```

上面的两行代码，都是右侧的运算数结合在一起。

另外，指数运算符（`**`）也是右结合。

```
2 ** 3 ** 2
// 相当于 2 ** (3 ** 2)
// 512
```

版本号 #2
由 壳壳虫 创建于 2018-01-11 18:02:27 UTC
由 壳壳虫 更新于 2020-05-25 17:53:19 UTC