扩展语句

扩展语法允许一个表达式在期望多个参数（用于函数调用）或多个元素（用于数组文本）或多个变量（用于解构赋值）的位置扩展。

语法

用于函数调用:

```
myFunction(...iterableObj);
```

用于数组字面量:

```
[...iterableObj, 4, 5, 6]
```

范例

更好的 apply 方法

在需要使用数组作为函数的参数的情况下,通常使用 `Function.prototype.apply` 方法:

```
function myFunction(x, y, z) { }
var args = [0, 1, 2];
myFunction.apply(null, args);
```

如果使用了ES6的展开运算符,你可以这么写:

```
function myFunction(x, y, z) { }
var args = [0, 1, 2];
myFunction(...args);
```

还可以同时展开多个数组:

```
function myFunction(v, w, x, y, z) { }
var args = [0, 1];
myFunction(-1, ...args, 2, ...[3]);
```

一个更强大的数组字面量

例子: 如果已经有一个数组，此时还需要再新建一个数组，要求新数组包含已有数组的数组项的话，就要用到push，splice，concat 等数组方法。有了扩展运算符会让代码更简洁:

```
let parts = ['shoulder', 'knees'];

let Tshirts = ['Lee', 'Nike'];


let lyrics = ['head', ...parts, 'and', 'toes'];

// ["head", "shoulders", "knees", "and", "toes"]



let lyrics = ['head', ...parts, 'and', 'toes', ...Tshirts];

// ["head", "shoulder", "knees", "and", "toes", "Lee", "Nike"]
```

就像扩展参数列表一样，...可以在数组字面量中的任何地方使用，可以多次使用。

配合new运算符

例子: 在ES5中,我们无法同时使用 new 运算符和 apply 方法(apply方法调用[[Call]]而不是[[Construct]])。在ES6中，我们可以使用扩展运算符，和普通的函数调用一样。

```
let dateFields = [1970, 0, 1];
// 1 Jan 1970
let d = new Date(...dateFields);


let dataFields = readDateFields(database);
let d = new Date(...dateFields);
```

复制一个数组

```
let arr = [1, 2, 3];
let arr2 = [...arr]; // like arr.slice()

arr2.push(4);

// arr2 becomes [1, 2, 3, 4]
// arr remains unaffected
```

Note: Spread syntax effectively goes one level deep while copying an array. Therefore, it may be unsuitable

for copying multidimensional arrays as the following example shows (it's the same with `Object.assign()` and spread syntax).

```
let a = [[1], [2], [3]];
let b = [...a];
b.shift().shift(); // 1
// Now array b is: [[2], [3]]
```

一个更好的连接数组的方法

例子: 在ES5中，我们通常使用 `push` 方法将一个数组添加到另一个数组的末尾:

```
let arr1 = [0, 1, 2];
let arr2 = [3, 4, 5];

// 将arr2中的所有元素添加到arr1中
Array.prototype.push.apply(arr1, arr2);
```

在ES6中，使用扩展运算符：

```
var arr1 = [0, 1, 2];
var arr2 = [3, 4, 5];

arr1.push(...arr2);
```

Example: `unshift` is often used to insert an array of values at the start of an existing array. Without spread syntax this is done as:

```
let arr1 = [0, 1, 2];
let arr2 = [3, 4, 5];

// Prepend all items from arr2 onto arr1

Array.prototype.unshift.apply(arr1, arr2);
// arr1 is now [3, 4, 5, 0, 1, 2]
```

With spread syntax this becomes:

```
let arr1 = [0, 1, 2];
let arr2 = [3, 4, 5];

arr1 = [...arr2, ...arr1];
// arr1 is now [3, 4, 5, 0, 1, 2]
```

仅可遍历对象(iterables)可用

```
let obj = {'key1': 'value1'};

let array = [...obj];
// TypeError: obj is not iterable




let obj = {"key1":"value1"};

function myFunction(x) {
    console.log(x);
    // undefined
}

myFunction(...obj);
let args = [...obj];

console.log(args, args.length);
//[] 0
```

将类数组对象转换成数组

扩展运算符可以将一个类数组对象中索引范围在[0,length)之间的所有属性的值添加到一个数组中,这样就可以得到一个真正的数组:

```
var nodeList = document.querySelectorAll('div');
var array = [...nodeList];
```

剩余操作符

还有一种操作符叫做剩余操作符（the rest operator），它的样子看起来和展开操作符一样，但是它是用于解构数组和对象。在某种程度上，剩余元素和展开元素相反，展开元素会“展开”数组变成多个元素，剩余元素会收集多个元素和“压缩”成一个单一的元素。

规范

规范	状态	备注
ECMAScript 2015 (6th Edition, ECMA-262)	ST Standard	规范中定义的几个部分: Array Initializer, Argument Lists
ECMAScript Latest Draft (ECMA-262)	D Draft	

浏览器兼容性

Desktop Mobile

Feature	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari (WebKit)
数组字面量中的展开操作	46	16 (16)	Edge	未实现	7.1
函数调用中的展开操作	46	27 (27)	Edge	未实现	7.1
解构赋值中的展开操作	49	34 (34)	未实现	?	?

相关链接

- 剩余参数
- 剩余操作符