



- 首页
- 所有文章
- JavaScript
- HTML5
- CSS
- 基础技术
- 职场
- 工具资源
- 前端小组
- 更多频道

- 导航条 -

伯乐在线 > WEB前端 - 伯乐在线 > 所有文章 > JavaScript > ES6 的 12 个核心功能一览

# ES6 的 12 个核心功能一览

2016/11/17 · JavaScript · 1 评论 · es6

分享到： 本文由 伯乐在线 - 古鲁伊 翻译。未经许可，禁止转载！  
英文出处：[Adrian Mejia](#)。欢迎加入[翻译组](#)。

过去几年 JavaScript 发生了很大的变化。下面的 12 个新功能现在就可以用起来了。

## JavaScript 历史

新补充的语言叫 ECMAScript 6，也叫 ES6 或 ES2015+。

JavaScript 自 1995 年面世以来，一直在缓慢地改进着。每隔几年都会有新的补充。1997 年成立的 ECMAScript 引领着 JavaScript 的发展。已发布的版本有 ES3、ES5、ES6 等。



## JavaScript 进化史

ES3 与 ES5 之间隔了 10 年，而 ES5 与 ES6 之间隔了 6 年。改进的新模式是每年都渐进式地做一些小改动，而不是像 ES6 一样一次性地进行大量的更改。

浏览器支持

所有的现代浏览器和环境都已经支持 ES6 了。

Feature name	Current browser	iOS 10	SF 10	Node 6.5 <sup>[6]</sup>	CH 54, OP 41 <sup>[11]</sup>	XS6	Edge 14 <sup>[4]</sup>	FF 49	FF 45 ESR	Edge 13 <sup>[4]</sup>	Babel + core-js <sup>[2]</sup>	JXA	Type-Script + core-js	T
Syntax														
● <a href="#">default function parameters</a>	▶	7/7	7/7	7/7	7/7	7/7	7/7	4/7	4/7	0/7	4/7	0/7	5/7	
● <a href="#">rest parameters</a>	▶	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	3/5	0/5	4/5	
● <a href="#">spread (...) operator</a>	▶	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	13/15	11/15	4/15	
● <a href="#">object literal extensions</a>	▶	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	5/6	6/6	
● <a href="#">for..of loops</a>	▶	9/9	9/9	9/9	9/9	9/9	7/9	7/9	7/9	7/9	9/9	8/9	3/9	
● <a href="#">octal and binary literals</a>	▶	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	
● <a href="#">template literals</a>	▶	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	4/5	5/5	3/5	
● <a href="#">RegExp "y" and "u" flags</a>	▶	5/5	5/5	5/5	5/5	5/5	2/5	5/5	2/5	5/5	3/5	0/5	0/5	
● <a href="#">destructuring, declarations</a>	▶	22/22	22/22	22/22	22/22	21/22	21/22	21/22	19/22	0/22	21/22	19/22	15/22	
● <a href="#">destructuring, assignment</a>	▶	24/24	24/24	24/24	24/24	24/24	23/24	23/24	21/24	0/24	24/24	21/24	19/24	
● <a href="#">destructuring, parameters</a>	▶	23/23	23/23	23/23	23/23	23/23	22/23	19/23	18/23	0/23	20/23	18/23	15/23	
● <a href="#">Unicode code point escapes</a>	▶	2/2	2/2	2/2	2/2	2/2	2/2	1/2	1/2	2/2	1/2	2/2	1/2	
● <a href="#">new.target</a>	▶	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	1/2	0/2	0/2	0/2	
Bindings														
● <a href="#">const</a>	▶	16/16	16/16	16/16	16/16	16/16	16/16	12/16	12/16	12/16	14/16	10/16	14/16	
● <a href="#">let</a>	▶	12/12	12/12	12/12	12/12	12/12	12/12	10/12	10/12	10/12	10/12	0/12	10/12	
● <a href="#">block-level function declaration</a> <sup>[13]</sup>	⊖	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	No	No	
Functions														
● <a href="#">arrow functions</a>	▶	13/13	13/13	13/13	13/13	12/13	13/13	13/13	13/13	13/13	9/13	0/13	9/13	
● <a href="#">class</a>	▶	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	19/24	18/24	19/24	
● <a href="#">super</a>	▶	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	4/8	7/8	7/8	
● <a href="#">generators</a>	▶	27/27	27/27	27/27	27/27	27/27	27/27	25/27	25/27	27/27	24/27	0/27	0/27	

来源：<https://kangax.github.io/compat-table/es6/>

Chrome、MS Edge、Firefox、Safari、Node和其它很多环境都已经嵌入程序以支持 JavaScript ES6 的大部分功能。所以从本教程学到的所有功能都可以直接使用。来开始学习 ECMAScript 6 吧！

## ES6 核心功能

可以在浏览器控制台上尝试下面的代码。

🔍

📄

Elements

Console

Sources

Network

Timeline

Profiles

Application

Security

Audi

🔇

🔊

top

▼

☐ Preserve log

> class Animal {  
 constructor(name) {  
 this.name = name;  
 }  
 speak() {  
 console.log(this.name + ' makes a noise.'); } }  
var animal = new Animal('animal');

animal.speak();  
animal makes a noise.

< undefined

> |

所以不要照单全收，试试每个 ES5 和 ES6 示例。我们开始吧。

### 块级作用域变量

在 ES6 中，我们使用 let/const，而非 var 来声明变量。

var 有什么缺点呢？

test(false) 应该返回 outer，但并不是，得到的值是 undefined。

为什么？

因为即使 if 块没有执行，第 4 行的表达式 var x 也被提升了。

var 提升：

- var 是在函数作用域中的，即使在声明前，它在整个函数内都是可用的。
- 声明被提升了。所以变量在声明前就可以使用。
- 初始化 没有 被提升。如果使用 var 的话，一定 要在顶部声明变量。
- 应用提升规则后，就好理解代码是如何执行的了：

```
1 ES5
2
3 var x = 'outer';
4 function test(inner) {
5   var x; // HOISTED DECLARATION
6   if (inner) {
7     x = 'inner'; // INITIALIZATION NOT HOISTED
8     return x;
9   }
10  return x;
11 }
```

ECMAScript 2015帮助解决了这个问题：

```
1 ES6
2
3 let x = 'outer';
4 function test(inner) {
5   if (inner) {
6     let x = 'inner';
7     return x;
8   }
9   return x; // gets result from line 1 as expected
10 }
11 test(false); // outer
12 test(true); // inner
```

用 let 替代 var 以便按预期的那样去执行代码。如果没有调用 if 块，变量 x 就不会提升到块外。

Let 提升 和 “temporal dead zone”

- 在 ES6 中，let 会将变量提升到块顶部（而 非 像 ES5 是函数顶部）。
- 但是在变量声明前引用变量会造成 ReferenceError。
- let 是块作用域的，不可以在声明前使用。
- “Temporal dead zone” 是块开始直到变量被声明的这段区域。

## IIFE

在介绍 IIFE 之前，我们来看个例子：

```
1 ES5
2
3 {
4   var private = 1;
5 }
6 console.log(private); // 1
```

如你所见，private 会发生泄漏。需要使用 IIFE（立即执行函数表达式）将其包起来：

```
1 ES5
2
3 (function(){
4   var private2 = 1;
5 })();
6 console.log(private2); // Uncaught ReferenceError
```

如果看过 jQuery/lodash 或是其它开源项目的代码，你会注意到它们都利用了 IIFE，以避免污染全局环境，而只在全局下定义 \_、\$或是 jQuery。

ES6 更工整，不再需要使用 IIFE，只要用块和 let 就可以了：

```
1 ES6
2
3 {
4   let private3 = 1;
5 }
6 console.log(private3); // Uncaught ReferenceError
```

## Const

如果不希望变量的值再改变，可以使用 const。

```
<> undefined

> x=2

Uncaught TypeError: Assignment to constant variable. (...)

>
```

总之：用 `let` 和 `const` 代替 `var`。

使用 `const` 进行引用；避免使用 `var`。

如果必须重新指定引用，可以用 `let` 代替 `const`。

### 文本模板

遇到文本模板时，不必再用嵌套连接了。比如：

```
1 ES5
2
3 var first = 'Adrian';
4 var last = 'Mejia';
5 console.log('Your name is ' + first + ' ' + last + '.');
```

现在可以用 反引号 ( ``` ) 和字符串插值 `${}`：

```
1 ES6
2
3 const first = 'Adrian';
4 const last = 'Mejia';
5 console.log(`Your name is ${first} ${last}.`);
```

### 多行字符串

不必像这样再连接 + n 字符串了：

```
1 ES5
2
3 var template = '<li *ngFor="let todo of todos" [ngClass]="{completed: todo.isDone}" >n' +
4 '  <div class="view">n' +
5 '    <input class="toggle" type="checkbox" [checked]="todo.isDone">n' +
6 '    <label></label>n' +
7 '    <button class="destroy"></button>n' +
8 '  </div>n' +
9 '  <input class="edit" value="">n' +
10 '</li>';
11 console.log(template);
```

ES6 中同样可以用反引号解决：

```
1 ES6
2
3 const template = `<li *ngFor="let todo of todos" [ngClass]="{completed: todo.isDone}" >
4   <div class="view">
5     <input class="toggle" type="checkbox" [checked]="todo.isDone">
6     <label></label>
7     <button class="destroy"></button>
8   </div>
9   <input class="edit" value="">
10 </li>`;
11 console.log(template);
```

两段代码会得到完全相同的结果。

### 解构赋值

ES6 解构非常简明并且好用。看看下面的例子：

#### 获取数组元素

```
1 ES5
2
3 var array = [1, 2, 3, 4];
4 var first = array[0];
5 var third = array[2];
6 console.log(first, third); // 1 3
```

等同于：

```
1 ES6
```

### 调换值

```
1 ES5
2
3 var a = 1;
4 var b = 2;
5 var tmp = a;
6 a = b;
7 b = tmp;
8 console.log(a, b); // 2 1
```

等同于

```
1 ES6
2
3 let a = 1;
4 let b = 2;
5 [a, b] = [b, a];
6 console.log(a, b); // 2 1
```

### 返回多个值的解构

```
1 ES5
2
3 function margin() {
4   var left=1, right=2, top=3, bottom=4;
5   return { left: left, right: right, top: top, bottom: bottom };
6 }
7 var data = margin();
8 var left = data.left;
9 var bottom = data.bottom;
10 console.log(left, bottom); // 1 4
```

在第3行，也可以像这样用数组返回（并保存序列）：

```
1 return [left, right, top, bottom];
```

但之后调用时需要考虑返回数据的顺序。

```
1 var left = data[0];
2 var bottom = data[3];
```

ES6 中调用时只会选择需要的数据（第 6 行）：

```
1 ES6
2
3 function margin() {
4   const left=1, right=2, top=3, bottom=4;
5   return { left, right, top, bottom };
6 }
7 const { left, bottom } = margin();
8 console.log(left, bottom); // 1 4
```

注意：第3行用用到了一些其它的 ES6 功能。可以将 { left: left } 简化为 { left }。看看和 ES5 的版本相比，现在多简洁啊~很酷不是吗？

### 参数匹配解构

```
1 ES5
2
3 var user = {firstName: 'Adrian', lastName: 'Mejia'};
4 function getFullName(user) {
5   var firstName = user.firstName;
6   var lastName = user.lastName;
7   return firstName + ' ' + lastName;
8 }
9 console.log(getFullName(user)); // Adrian Mejia
```

等同于（但更简洁）：

```
1 ES6
2
3 const user = {firstName: 'Adrian', lastName: 'Mejia'};
4 function getFullName({ firstName, lastName }) {
5   return `${firstName} ${lastName}`;
6 }
7 console.log(getFullName(user)); // Adrian Mejia
```

### 深度匹配

首页 资讯 文章 资源 小组 相亲

频道 登录 注册

1

ES5

2

3

function settings() {

4

return { display: { color: 'red' }, keyboard: { layout: 'querty' } };

5

}

6

var tmp = settings();

7

var displayColor = tmp.display.color;

8

var keyboardLayout = tmp.keyboard.layout;

9

console.log(displayColor, keyboardLayout); // red querty

等同于（但更简洁）：

ES6

function settings() {

return { display: { color: 'red' }, keyboard: { layout: 'querty' } };

}

const { display: { color: displayColor }, keyboard: { layout: keyboardLayout } } = settings();

console.log(displayColor, keyboardLayout); // red querty

也叫对象解构。

如你所见，解构很有用，并有助于形成好的编码风格。

最佳实践：

- 使用数组解构获取元素或调换变量，这样就不用创建临时引用了。
- 对于多返回值的情况，不要用数组解构，用对象解构。

## 类和对象

ES6 用“类”替代“构造函数”。

在 JavaScript 中，每个对象都有原型对象。所有 JavaScript 对象都从原型上继承方法和属性。

ES5 以面向对象编程（OOP）的方式创建对象，是利用构造函数实现的：

ES5

var Animal = (function () {

function MyConstructor(name) {

this.name = name;

}

MyConstructor.prototype.speak = function speak() {

console.log(this.name + ' makes a noise.');

};

return MyConstructor;

})();

var animal = new Animal('animal');

animal.speak(); // animal makes a noise.

ES6 提供了语法糖，可以用 `class`、`constructor` 等新的关键字、更少的样板代码实现相同的效果。同样可以看到相比于 `constructor.prototype.speak = function ()`，用 `speak()` 定义方法更加清晰：

ES6

class Animal {

constructor(name) {

this.name = name;

}

speak() {

console.log(this.name + ' makes a noise.');

}

}

const animal = new Animal('animal');

animal.speak(); // animal makes a noise.

可以看到两种方式（ES5/6）的结果和使用方式相同。

最佳实践：

- 最好使用 `class` 语法，避免直接操作 `prototype`。原因是这样代码更加简明易懂。
- 避免出现空的构造器。如果没有指明，类会有默认的构造器的。

## 继承

基于前面的 `Animal` 类，现在想要拓展 `Animal`，定义一个 `Lion` 类。

ES5 原型继承的方式有些复杂。

ES5

var Lion = (function () {

function MyConstructor(name){

Animal.call(this, name);

}

// prototypal inheritance

MyConstructor.prototype = Object.create(Animal.prototype);

MyConstructor.prototype.constructor = Animal;

```
12 console.log(this.name + ' roars ');
13 };
14 return MyConstructor;
15 }());
16 var lion = new Lion('Simba');
17 lion.speak(); // Simba makes a noise.
18 // Simba roars.
```

在此我没有详细解读所有的代码，但是需要注意：

- 第 3 行，明确地用参数调用 `Animal` 构造器。
- 7-8 行，将 `Lion` 原型赋值为 `Animal` 的原型。
- 11 行，从父类 `Animal` 调用了 `speak` 方法。

ES6 提供了新的关键字 `extends` 和 `super`。

```
1 ES6
2
3 class Lion extends Animal {
4   speak() {
5     super.speak();
6     console.log(this.name + ' roars ');
7   }
8 }
9 const lion = new Lion('Simba');
10 lion.speak(); // Simba makes a noise.
11 // Simba roars.
```

效果相同，但是相比于 ES5，ES6 代码更易读，完胜。

最佳实践：

- 使用内置的 `extends` 实现继承。

## 原生 Promise

用 promise 替代回调地狱

```
1 ES5
2
3 function printAfterTimeout(string, timeout, done){
4   setTimeout(function(){
5     done(string);
6   }, timeout);
7 }
8 printAfterTimeout('Hello ', 2e3, function(result){
9   console.log(result);
10  // nested callback
11  printAfterTimeout(result + 'Reader', 2e3, function(result){
12    console.log(result);
13  });
14 });
```

这个函数接收一个回调，在 `done` 后执行。我们想要先后执行两次，所以在回调中又一次调用了 `printAfterTimeout`。

如果需要 3 或 4 次回调，代码很快就一团糟了。那么用 promise 如何实现呢：

```
1 ES6
2
3 function printAfterTimeout(string, timeout){
4   return new Promise((resolve, reject) => {
5     setTimeout(function(){
6       resolve(string);
7     }, timeout);
8   });
9 }
10 printAfterTimeout('Hello ', 2e3).then((result) => {
11   console.log(result);
12   return printAfterTimeout(result + 'Reader', 2e3);
13 }).then((result) => {
14   console.log(result);
15 });
```

promise 中可以用 `then` 在某个函数完成后执行新的代码，而不必再嵌套函数。

## 箭头函数

ES6 没有移除函数表达式，但是新增了箭头函数。

ES5 中，`this` 的指向有问题：

```
1 ES5
2
3 var _this = this; // need to hold a reference
4 $(' .btn').click(function(event){
5   _this.sendData(); // reference outer this
6 });
7 $(' .input').on('change', function(event){
8   this.sendData(); // reference outer this
9 }.bind(this)); // bind to outer this
```

## For...of

最开始用 `for` , 然后使用 `forEach` , 而现在可以用 `for...of` :

```
1 ES6
2
3 // this will reference the outer one
4 $('btn').click((event) => this.sendData());
5 // implicit returns
6 const ids = [291, 288, 984];
7 const messages = ids.map(value => `ID is ${value}`);
```

ES6 的 `for...of` 也可以用来迭代。

## 默认参数

之前需要检测变量是否定义了, 而现在可以指定 `default parameters` 的值。或许你之前像下面这样写过？

```
1 ES5
2
3 function point(x, y, isFlag){
4   x = x || 0;
5   y = y || -1;
6   isFlag = isFlag || true;
7   console.log(x,y, isFlag);
8 }
9 point(0, 0) // 0 -1 true
10 point(0, 0, false) // 0 -1 true
11 point(1) // 1 -1 true
12 point() // 0 -1 true
```

这可能是检测变量有值或指定默认值的惯用模式, 但也存在一些问题：

- 第 8 行, 我们传的值是 `0, 0` 但是得到的是 `0, -1`
- 第 9 行, 传进去 `false` 但是得到的是 `true`。

如果默认参数是布尔值或将值设为 `0`, 是没有用的。想知道为什么？我会在下面的 ES6 示例后说明。

有了 ES6, 现在可以用更少的代码实现更好的效果了。

```
1 ES6
2
3 function point(x = 0, y = -1, isFlag = true){
4   console.log(x,y, isFlag);
5 }
6 point(0, 0) // 0 0 true
7 point(0, 0, false) // 0 0 false
8 point(1) // 1 -1 true
9 point() // 0 -1 true
```

注意第 5 行和第 6 行我们拿到了想要的值。ES5 的示例不好用, 是因为先要检测 `undefined` 的值, 而 `false`、`null`、`undefined` 和 `0` 都是假的值。我们可以加些代码：

```
1 ES5
2
3 function point(x, y, isFlag){
4   x = x || 0;
5   y = typeof(y) === 'undefined' ? -1 : y;
6   isFlag = typeof(isFlag) === 'undefined' ? true : isFlag;
7   console.log(x,y, isFlag);
8 }
9 point(0, 0) // 0 0 true
10 point(0, 0, false) // 0 0 false
11 point(1) // 1 -1 true
12 point() // 0 -1 true
```

现在当检测 `undefined` 值时就符合我们的要求了。

## 剩余参数

之前使用 `arguments` , 而现在可以用展开操作符。

ES5 中处理不定参数很麻烦：

```
1 ES5
2
3 function printf(format) {
4   var params = [].slice.call(arguments, 1);
5   console.log('params: ', params);
6   console.log('format: ', format);
7 }
8 printf('%s %d %.2f', 'adrian', 321, Math.PI);
```

现在可以用展开操作符 `...` 达到相同的目的。

```
1 ES6
```



## 展开操作符

之前用 `apply()`，现在可以方便地使用展开操作符 ... 了：

提示：`apply()` 可以将数组转化为一系列参数。例如 `Math.max()` 接收一系列参数，但如果想应用于数组的话可以用 `apply` 帮助实现。

>

Math.max(2,100,1,6,43)

<

100

>

Math.max([2,100,1,6,43])

<

NaN

>

Math.max.apply(Math,[2,100,1,6,43])

<

100

如上所述，`apply` 可以将数组当作参数序列进行传递：

1

ES5

2

3

Math.max.apply(Math,[2,100,1,6,43]) // 100

ES6 可以用展开操作符：

1

ES6

2

3

Math.max(...[2,100,1,6,43]) // 100

之前用 `concat` 合并数组，现在也可以用展开操作符：

1

ES5

2

3

var array1 = [2,100,1,6,43];

4

var array2 = ['a', 'b', 'c', 'd'];

5

var array3 = [false, true, null, undefined];

6

console.log(array1.concat(array2, array3));

ES6 可以用展开操作符展开嵌套的数组：

1

ES6

2

3

const array1 = [2,100,1,6,43];

4

const array2 = ['a', 'b', 'c', 'd'];

5

const array3 = [false, true, null, undefined];

6

console.log([...array1, ...array2, ...array3]);

## 总结

JavaScript 已经发生了许多改变。本文包含了大部分的核心功能，这些是每个 JavaScript 程序员都应该知道的。本文还涉及了一些最佳实践，可以使你的代码更加简明易懂。

如果你觉得还有一些其它的必会功能，请在下方留言，方便我更新本文。



打赏支持我翻译更多好文章，谢谢！

¥ 打赏译者


👍 2 赞

🔖 5 收藏

💬 1 评论



关于作者：古鲁伊



立志做一名有格调的程序媛

👤 个人主页 · 📁 我的文章 · 🎓 22

### 相关文章

- ES6 你可能不知道的事 – 进阶篇
- 浅谈ES6原生Promise
- 如果了解 npm 和 ES6 模块，那就看过来 · 1
- ES6 你可能不知道的事 – 基础篇

可能感兴趣的话题


- [你愿意去到对方的城市发展吗？ · 1](#)
- [Java Applet与Javascript的通信](#)
- [jQuery对象和原生DOM对象的相互转换以及实现原理](#)
- [程序员的直觉 · 5](#)
- [安装Oracle 11g问题](#)
- [AndroidLinker与SO加壳技术的关系](#)

登录后评论

新用户注册

直接登录

最新评论



飘逸、麦子 ( 1 )

开发

6 天前

赞一个！

赞 回复

前端小组话题

我有新话题



[好纠结！要不要辞职去上前端培训班](#)  
[hai^Q^](#) 发起 • 104 回复



[jQuery对象和原生DOM对象的相互...](#)  
[ice\\_shou](#) 发起



[前端培训这个是坑么，帮我看](#)  
[qt1144](#) 发起 • 23 回复



[有前端同学所在公司使用了typescript...](#)  
[sheldon shen](#) 发起 • 7 回复



[前端自学真的很难找工作吗啊？](#)  
[一抹茶](#) 发起 • 12 回复



[本人大三前端专业，每次写完代码写...](#)  
[yumi](#) 发起 • 58 回复



- [本周热门前端文章](#)
- [本月热门](#)
- [热门标签](#)

- [前端面试题整理汇总](#)
- [ES6 的 12 个核心功能一览](#)
- [远离面条代码：编写可维护的 JS ...](#)
- [关于 Web 安全，99% 的网站都忽...](#)
- [实现前端资源增量式更新的一种思路](#)

- 6 [到底vuex是什么？](#)
- 7 [Vue.js 和 MVVM 的小细节](#)
- 8 [引人瞩目的 CSS 变量](#)
- 9 [番茄色 VS #FF6347——CSS...](#)



业界热点资讯

更多 »



[谷歌工程师：杀毒软件根本没什么用](#)

2 天前 · 29 · 5



[中国超级计算机再称霸 美媒：特朗普不会允许美国当第二](#)

1 天前 · 4



[宽松开源许可证的崛起意味着什么？](#)

2 天前 · 6



[IBM、Intel、谷歌、微软这些巨头们都在干这事](#)

2 天前 · 7



[微软再发力，正式宣布开源 JDBC 驱动程序](#)

2 天前 · 9



[Linux 爆新漏洞，长按回车键70秒即可获得root权限](#)

5 天前 · 62 · 5

前端工具资源

更多资源 »



[Velocity.js：加速JavaScript动画](#)

[动画](#)



[three.js：JavaScript 3D 库](#)

[JavaScript](#), [Web 数据可视化工具](#)



[jquery.transit：提供流畅CSS3变换和过渡效果的jQuery插件](#)

[JavaScript](#), [动画](#)



[bounce.js：创建有趣的CSS3动画](#)

[JavaScript](#), [动画](#)

[lodash：模块化、高性能实用工具库](#)  
[JavaScript, 函数式编程](#) · [1](#)

关于伯乐前端

伯乐前端分享Web前端开发，包括JavaScript，CSS和HTML5开发技术，前端相关的行业动态。

快速链接

- [网站使用指南](#) »
- [加入我们](#) »
- [问题反馈与求助](#) »
- [网站积分规则](#) »
- [网站声望规则](#) »

关注我们

新浪微博：[@前端大全](#)  
RSS：[订阅地址](#)

推荐微信号



合作联系

Email：[bd@jobbole.com](mailto:bd@jobbole.com)  
QQ：2302462408（加好友请注明来意）

更多频道

- [小组](#) – 好的话题、有启发的回复、值得信赖的圈子
- [头条](#) – 分享和发现有价值的内容与观点
- [相亲](#) – 为IT单身男女服务的征婚传播平台
- [资源](#) – 优秀的工具资源导航
- [翻译](#) – 翻译传播优秀的外文文章
- [文章](#) – 国内外的精选文章
- [设计](#) – UI,网页，交互和用户体验
- [iOS](#) – 专注iOS技术分享
- [安卓](#) – 专注Android技术分享
- [前端](#) – JavaScript, HTML5, CSS
- [Java](#) – 专注Java技术分享
- [Python](#) – 专注Python技术分享

