

Object.defineProperty()

Object.defineProperty() 方法会直接在一个对象上定义一个新属性，或者修改一个已经存在的属性，并返回这个对象。

语法

```
Object.defineProperty(obj, prop, descriptor)
```

参数

obj

需要定义属性的对象。

prop

需定义或修改的属性的名字。

descriptor

将被定义或修改的属性的描述符。

返回值

返回传入函数的对象，即第一个参数obj

描述

该方法允许精确添加或修改对象的属性。一般情况下，我们为对象添加属性是通过赋值来创建并显示在属性枚举中（[for...in](#) 或 [Object.keys](#) 方法），但这种方式添加的属性值可以被改变，也可以被删除。而使用 [Object.defineProperty\(\)](#) 则允许改变这些额外细节的默认设置。例如，默认情况下，使用 [Object.defineProperty\(\)](#) 增加的属性值是不可改变的。

对象里目前存在的属性描述符有两种主要形式：数据描述符和存取描述符。数据描述符是一个拥有可写或不可写值的属性。存取描述符是由一对 [getter-setter](#) 函数功能来描述的属性。描述符必须是两种形式之一；不能同时是两者。

数据描述符和存取描述符均具有以下可选键值：

configurable

当且仅当该属性的 `configurable` 为 `true` 时，该属性才能够被改变，也能够被删除。默认为 **false**。

enumerable

当且仅当该属性的 `enumerable` 为 `true` 时，该属性才能够出现在对象的枚举属性中。默认为 **false**。

数据描述符同时具有以下可选键值：

value

该属性对应的值。可以是任何有效的 JavaScript 值（数值，对象，函数等）。默认为 **undefined**。

writable

当且仅当该属性的 `writable` 为 `true` 时，该属性才能被赋值运算符改变。默认为 **false**。

存取描述符同时具有以下可选键值：

get

一个给属性提供 getter 的方法，如果没有 getter 则为 `undefined`。该方法返回值被用作属性值。默认为 **undefined**。

set

一个给属性提供 setter 的方法，如果没有 setter 则为 `undefined`。该方法将接受唯一参数，并将该参数的新值分配给该属性。默认为 **undefined**。

记住，这些选项不一定是自身属性，如果是继承来的也要考虑。为了确认保留这些默认值，你可能要在这之前冻结 `Object.prototype`，明确指定所有的选项，或者将 `__proto__` 属性指向 `null`。

```
1 // 使用 __proto__
2 Object.defineProperty(obj, "key", {
3   __proto__: null, // 没有继承的属性
4   value: "static"  // 没有 enumerable
5                   // 没有 configurable
6                   // 没有 writable
7                   // 作为默认值
8 });
9
10 // 显式
11 Object.defineProperty(obj, "key", {
12   enumerable: false,
13   configurable: false,
14   writable: false,
15   value: "static"
16 });
```

```
17 // 循环使用同一对象
18 function withValue(value) {
19     var d = withValue.d || (
20         withValue.d = {
21             enumerable: false,
22             writable: false,
23             configurable: false,
24             value: null
25         }
26     );
27     d.value = value;
28     return d;
29 }
30 // ... 并且 ...
31 Object.defineProperty(obj, "key", withValue("static"));
32
33 // 如果 freeze 可用, 防止代码添加或删除对象原型的属性
34 // (value, get, set, enumerable, writable, configurable)
35 (Object.freeze || Object)(Object.prototype);
36
```

示例

如果你想知道如何用 *binary-flags-like* 语法使用 `Object.defineProperty` 方法, 看看[这篇文章](#)。

创建属性

如果对象中不存在指定的属性, `Object.defineProperty()` 就创建这个属性。当描述符中省略某些字段时, 这些字段将使用它们的默认值。拥有布尔值的字段的默认值都是 `false`。 `value`, `get` 和 `set` 字段的默认值为 `undefined`。定义属性时如果没有 `get/set/value/writable`, 那它被归类为数据描述符。

```
1 var o = {}; // 创建一个新对象
2
3 // Example of an object property added with defineProperty with a data pro
4 Object.defineProperty(o, "a", {value : 37,
5                               writable : true,
6                               enumerable : true,
7                               configurable : true});
8 // 对象o拥有了属性a, 值为37
9
10 // Example of an object property added with defineProperty with an access
11 var bValue;
12 Object.defineProperty(o, "b", {get : function(){ return bValue; },
13                               set : function(newValue){ bValue = newValu
14                               enumerable : true,
15                               configurable : true});
```

```
16 o.b = 38;
17 // 对象o拥有了属性b，值为38
18
19 // The value of o.b is now always identical to bValue, unless o.b is rede
20
21 // 数据描述符和存取描述符不能混合使用
22 Object.defineProperty(o, "conflict", { value: 0x9f91102,
23                                     get: function() { return 0xdeadbee
24 // throws a TypeError: value appears only in data descriptors, get appear
```

修改属性

如果属性已经存在，`Object.defineProperty()` 将尝试根据描述符中的值以及对象当前的配置来修改这个属性。如果描述符的 `configurable` 特性为 `false`（即该特性为 `non-configurable`），那么除了 `writable` 外，其他特性都不能被修改，并且数据和存取描述符也不能相互切换。

如果一个属性的 `configurable` 为 `false`，则其 `writable` 特性也只能修改为 `false`。

如果尝试修改 `non-configurable` 属性特性（除 `writable` 以外），将会产生一个 `TypeError` 异常，除非当前值与修改值相同。

Writable 属性

当属性特性（property attribute）`writable` 设置为 `false` 时，表示 `non-writable`，属性不能被修改。

```
1 var o = {}; // 创建一个新对象
2
3 Object.defineProperty(o, "a", { value : 37,
4                               writable : false });
5
6 console.log(o.a); // 打印 37
7 o.a = 25; // 没有错误抛出（在严格模式下会抛出，即使之前已经有相同的值）
8 console.log(o.a); // 打印 37， 赋值不起作用。
```

正如上例中看到的，修改一个 `non-writable` 的属性不会改变属性的值，同时也不会报异常。

Enumerable 特性

属性特性 `enumerable` 定义了对对象的属性是否可以在 `for...in` 循环和 `Object.keys()` 中被枚举。

```
1 var o = {};
2 Object.defineProperty(o, "a", { value : 1, enumerable:true });
3 Object.defineProperty(o, "b", { value : 2, enumerable:false });
```

```
4 Object.defineProperty(o, "c", { value : 3 }); // enumerable defaults to false
5 o.d = 4; // 如果使用直接赋值的方式创建对象的属性，则这个属性的enumerable为true
6
7 for (var i in o) {
8     console.log(i);
9 }
10 // 打印 'a' 和 'd' (in undefined order)
11
12 Object.keys(o); // ["a", "d"]
13
14 o.propertyIsEnumerable('a'); // true
15 o.propertyIsEnumerable('b'); // false
16 o.propertyIsEnumerable('c'); // false
```

Configurable 特性

configurable 特性表示对象的属性是否可以被删除，以及除 writable 特性外的其他特性是否可以被修改。

```
1 var o = {};
2 Object.defineProperty(o, "a", { get : function(){return 1;},
3                               configurable : false } );
4
5 // throws a TypeError
6 Object.defineProperty(o, "a", {configurable : true});
7 // throws a TypeError
8 Object.defineProperty(o, "a", {enumerable : true});
9 // throws a TypeError (set was undefined previously)
10 Object.defineProperty(o, "a", {set : function(){} });
11 // throws a TypeError (even though the new get does exactly the same thing)
12 Object.defineProperty(o, "a", {get : function(){return 1;}});
13 // throws a TypeError
14 Object.defineProperty(o, "a", {value : 12});
15
16 console.log(o.a); // logs 1
17 delete o.a; // Nothing happens
18 console.log(o.a); // logs 1
```

如果 o.a 的 configurable 特性已经为 true，没有错误会被抛出，并且属性会在最后被删除。

添加多个属性和默认值

考虑特性被赋予的默认特性值非常重要，通常，使用点运算符和Object.defineProperty()为对象的属性赋值时，数据描述符中的属性默认值是不同的，如下例所示。

```
1  var o = {};  
2  
3  o.a = 1;  
4  // 等同于 :  
5  Object.defineProperty(o, "a", {  
6    value : 1,  
7    writable : true,  
8    configurable : true,  
9    enumerable : true  
10 });  
11  
12  
13 // 另一方面,  
14 Object.defineProperty(o, "a", { value : 1 });  
15 // 等同于 :  
16 Object.defineProperty(o, "a", {  
17   value : 1,  
18   writable : false,  
19   configurable : false,  
20   enumerable : false  
21 });
```

一般的 Setters 和 Getters

下面的例子说明了如何实现自我存档的对象。当 `temperature` 属性设置时，`archive` 数组会得到一个 `log`。

```
1  function Archiver() {  
2    var temperature = null;  
3    var archive = [];  
4  
5    Object.defineProperty(this, 'temperature', {  
6      get: function() {  
7        console.log('get!');  
8        return temperature;  
9      },  
10     set: function(value) {  
11       temperature = value;  
12       archive.push({ val: temperature });  
13     }  
14   });  
15  
16   this.getArchive = function() { return archive; };  
17 }
```

```
18
19 var arc = new Archiver();
20 arc.temperature; // 'get!'
21 arc.temperature = 11;
22 arc.temperature = 13;
23 arc.getArchive(); // [{ val: 11 }, { val: 13 }]
```

另一个例子：

```
1  var pattern = {
2      get: function () {
3          return 'I always return this string,whatever you have assigned';
4      },
5      set: function () {
6          this.myname = 'this is my name string';
7      }
8  };
9
10
11 function TestDefineSetAndGet() {
12     Object.defineProperty(this, 'myproperty', pattern);
13 }
14
15
16 var instance = new TestDefineSetAndGet();
17 instance.myproperty = 'test';
18
19 // 'I always return this string,whatever you have assigned'
20 console.log(instance.myproperty);
21 // 'this is my name string'
22 console.log(instance.myname);
```

规范

规范版本	规范状态	说明
ECMAScript 5.1 (ECMA-262) Object.defineProperty	ST Standard	Initial definition. Implemented in JavaScript 1.8.5
ECMAScript 2015 (6th Edition, ECMA-262) Object.defineProperty	ST Standard	
ECMAScript 2017 Draft (ECMA-262) Object.defineProperty	D Draft	

浏览器支持

Desktop

Mobile

特性	Firefox (Gecko)	Chrome	Internet Explorer	Opera	Safari
基本支持	4.0 (2)	5	9 [1]	11.60	5.1 [2]

[1] 在IE8中只支持 DOM 对象，同时也存在一些非标准的行为。

[2] Safari 5中也支持，但不能是 DOM 对象。

兼容性问题

重定义数组对象的 length 属性

数组的 `length` 属性重定义是可能的，但是会受到一般的重定义限制。（`length` 属性初始为 `non-configurable`，`non-enumerable` 以及 `writable`。对于一个内容不变的数组，改变其 `length` 属性的值或者使它变为 `non-writable` 是可能的。但是改变其可枚举性和可配置性或者当它是 `non-writable` 时尝试改变它的值或是可写性，这两者都是不允许的。）然而，并不是所有的浏览器都允许 `Array.length` 的重定义。

在 Firefox 4 至 22 版本中尝试去重定义数组的 `length` 属性都会抛出一个 `TypeError` 异常。

有些版本的Chrome中，`Object.defineProperty()` 在某些情况下会忽略不同于数组当前`length`属性的`length`值。有些情况下改变可写性并不起作用（也不抛出异常）。同时，比如`Array.prototype.push`的一些数组操作方法也不会考虑不可读的`length`属性。

有些版本的Safari中，`Object.defineProperty()` 在某些情况下会忽略不同于数组当前`length`属性的`length`值。尝试改变可写性的操作会正常执行而不抛出错误，但事实上并未改变属性的可写性。

只在Internet Explorer 9及以后版本和Firefox 23及以后版本中，才完整地正确地支持数组`length`属性的重新定义。目前不要依赖于重定义数组`length` 属性能够起作用，或在特定情形下起作用。与此同时，即使你能够依赖于它，你也[没有合适的理由这样做](#)。

Internet Explorer 8 具体案例

Internet Explorer 8 实现了 `Object.defineProperty()` 方法，但[只能在 DOM 对象上使用](#)。需要注意的一些事情：

- 尝试在原生对象上使用 `Object.defineProperty()` 会报错。
- 属性特性必须设置一些值。数据描述符为 `true`，`true`，`true`，`configurable` 为 `true`，`enumerable` 和 `accessor` 描述符为 `false`。(?) 任何试图提供其他值(?)将导致一个错误抛出。
- 重新配置一个属性首先需要删除该属性。如果属性没有删除，就如同重新配置前的尝试。

相关链接

- [Enumerability and ownership of properties](#)
- [Object.defineProperties\(\)](#)
- [Object.propertyIsEnumerable\(\)](#)
- [Object.getOwnPropertyDescriptor\(\)](#)
- [Object.prototype.watch\(\)](#)
- [Object.prototype.unwatch\(\)](#)
- [get](#)
- [set](#)
- [Object.create\(\)](#)
- [Additional Object.defineProperty examples](#)
- [Reflect.defineProperty\(\)](#)