Object.defineProperty()

`Object.defineProperty()` 方法会直接在一个对象上定义一个新属性，或者修改一个对象的现有属性， 并返回这个对象。

语法

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

参数

`obj`
需要被操作的目标对象

`prop`
目标对象需要定义或修改的属性的名称。

`descriptor`
将被定义或修改的属性的描述符。

返回值

被传递给函数的对象。

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 描述符 才能够被改变，同时该属性也能从对应的对象上被删除。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` 。

```
// 使用 __proto__
 var obj = {};
var descriptor = Object.create(null); //  没有继承的属性
// 默认没有 enumerable，没有 configurable，没有 writable
descriptor.value = 'static';
Object.defineProperty(obj, 'key', descriptor);

// 显式
Object.defineProperty(obj, "key", {
  enumerable: false,
  configurable: false,
  writable: false,
  value: "static"
});

// 循环使用同一对象
```

```
function withValue(value) {
  var d = withValue.d || (
    withValue.d = {
      enumerable: false,
      writable: false,
      configurable: false,
      value: null
    }
  );
  d.value = value;
  return d;
}
// ... 并且 ...
Object.defineProperty(obj, "key", withValue("static"));

// 如果 freeze 可用，防止代码添加或删除对象原型的属性
// （value, get, set, enumerable, writable, configurable）
(Object.freeze||Object)(Object.prototype);
```

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。

```
var o = {}; // 创建一个新对象

// 在对象中添加一个属性与数据描述符的示例
Object.defineProperty(o, "a", {
  value : 37,
  writable : true,
  enumerable : true,
  configurable : true
});

// 对象o拥有了属性a，值为37

// 在对象中添加一个属性与存取描述符的示例
var bValue;
Object.defineProperty(o, "b", {
  get : function(){
```

```
    return bValue;
  },
  set : function(newValue){
    bValue = newValue;
  },
  enumerable : true,
  configurable : true
});

o.b = 38;
// 对象o拥有了属性b，值为38

// o.b的值现在总是与bValue相同，除非重新定义o.b

// 数据描述符和存取描述符不能混合使用
Object.defineProperty(o, "conflict", {
  value: 0x9f91102,
  get: function() {
    return 0xdeadbeef;
  }
});
// 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。

```
var o = {}; // 创建一个对象

Object.defineProperty(o, "a", { value : 37,
```

```
                                    writable : false });

console.log(o.a); // 打印 37
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）
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` 定义了对象的属性是否可以在 `for...in` 循环和 `Object.keys()` 中被枚举。

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

for (var i in o) {
  console.log(i);
}
// 打印 'a' 和 'd' (in undefined order)

Object.keys(o); // ["a", "d"]

o.propertyIsEnumerable('a'); // true
o.propertyIsEnumerable('b'); // false
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。

```
var o = {};
Object.defineProperty(o, "a", { get : function(){return 1;},
                                configurable : false } );

// throws a TypeError
Object.defineProperty(o, "a", {configurable : true});
```

```
// throws a TypeError
Object.defineProperty(o, "a", {enumerable : true});
// throws a TypeError (set was undefined previously)
Object.defineProperty(o, "a", {set : function(){}});
// throws a TypeError (even though the new get does exactly the same thir
Object.defineProperty(o, "a", {get : function(){return 1;}});
// throws a TypeError
Object.defineProperty(o, "a", {value : 12});

console.log(o.a); // logs 1
delete o.a; // Nothing happens
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。

```
var o = {};

o.a = 1;
// 等同于 :
Object.defineProperty(o, "a", {
  value : 1,
  writable : true,
  configurable : true,
  enumerable : true
});


// 另一方面，
Object.defineProperty(o, "a", { value : 1 });
// 等同于 :
Object.defineProperty(o, "a", {
  value : 1,
  writable : false,
  configurable : false,
  enumerable : false
});
```

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。

```
function Archiver() {
  var temperature = null;
  var archive = [];

  Object.defineProperty(this, 'temperature', {
    get: function() {
      console.log('get!');
      return temperature;
    },
    set: function(value) {
      temperature = value;
      archive.push({ val: temperature });
    }
  });

  this.getArchive = function() { return archive; };
}

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

另一个例子：

```
var pattern = {
    get: function () {
        return 'I alway return this string,whatever you have assigned';
    },
    set: function () {
        this.myname = 'this is my name string';
    }
};


```

```
11  function TestDefineSetAndGet() {
12      Object.defineProperty(this, 'myproperty', pattern);
13  }
14
15
16  var instance = new TestDefineSetAndGet();
17  instance.myproperty = 'test';
18
19  // 'I alway 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 Latest Draft (ECMA-262) Object.defineProperty	LS Living Standard	

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特定的值。对于数据属性描述符， `configurable` , `enumerable` 和 `writable` 特性必须全部设置为 `true`； 对于访问器属性描述符， `configurable` 必须设置为 `true` ， `enumerable` 必须设置为 `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()`