Object 对象

《JavaScript 教程》作者：阮一峰

1 概述

JavaScript 的所有其他对象都继承自 Object对象，即那些对象都是Object的实例。

Object对象的原生方法分成两类：

- **静态方法** 是直接定义在 Object 对象上的方法。
- **实例方法** 是定义在原型对象 Object.prototype 上的方法。它可以被 Object实例 直接使用。

2 Object 构造函数

Object 构造函数是使用 new 关键字来生成新对象。

语法：

```
var obj1 = new Object();
```

注意：下面的写法与上面是等效的，都创建了一个新的空对象，下面的写法只是简便写法而已。

```
var obj2 = {};
```

传入参数：

对象或原始类型值。

返回值：

1. 如果参数是一个对象，则 **直接返回这个对象。**

   ```
   var o1 = {a: 1};
   var o2 = new Object(o1);
   o1 === o2 // true
   ```

2. 如果参数是一个原始类型的值，则返回对应的 **包装对象的实例。**

   ```
   var obj = new Object(123);
   obj instanceof Number // true
   ```

工具方法 Object()

Object() 当作工具方法使用，可以将任意值转为对象。

这个方法常用于保证某个值一定是对象。

工具方法的传入参数与返回值：

1. 如果 **传入参数为空**，则返回 **空对象。**

```
var obj = Object();
// 等同于
var obj = Object(undefined);
var obj = Object(null);

obj instanceof Object // true
```

2. 如果参数是**原始类型值**，则将其转为对应的 **包装对象的实例。**

```
var obj = Object(1);
obj instanceof Object // true
obj instanceof Number // true

var obj = Object('foo');
obj instanceof Object // true
obj instanceof String // true

var obj = Object(true);
obj instanceof Object // true
obj instanceof Boolean // true
```

3. 如果传入参数是一个**对象**，不进行转换，**直接返回该对象。**

```
var arr = [];
var obj = Object(arr); // 返回原数组
obj === arr // true

var value = {};
var obj = Object(value) // 返回原对象
obj === value // true

var fn = function () {};
var obj = Object(fn); // 返回原函数
obj === fn // true
```

利用这一点，可以写一个判断变量是否为对象的函数。

```
function isObject(value) {
  return value === Object(value);
}

isObject([]) // true
isObject(true) // false
```

3 Object 的静态方法

1. 遍历对象属性的方法

一般情况下，几乎总是使用Object.keys方法，遍历对象的属性。

- **Object.keys()**

  传入参数：对象。

  返回值：数组。数组成员为该对象自身的所有属性名，不包括继承的属性名。

```
var obj = {
  p1: 123,
  p2: 456
};

Object.keys(obj) // ["p1", "p2"]
```

- **Object.getOwnPropertyNames()**

  传入参数：对象。

  返回值：数组。数组成员为该对象自身的所有属性名，不包括继承的属性名。

```
var obj = {
  p1: 123,
  p2: 456
};

Object.getOwnPropertyNames(obj) // ["p1", "p2"]
```

  二者的区别

  对于一般的对象来说，Object.keys()和Object.getOwnPropertyNames()返回的结果是一样的。只有涉及不可枚举属性时，才会有不一样的结果。Object.keys方法只返回可枚举的属性，Object.getOwnPropertyNames方法还返回不可枚举的属性名。

```
var a = ['Hello', 'World'];

Object.keys(a) // ["0", "1"]
Object.getOwnPropertyNames(a) // ["0", "1", "length"]
```

  上面代码中，数组的 length 属性是不可枚举的属性，所以只出现在 Object.getOwnPropertyNames 方法的返回结果中。

2. 对象属性模型的相关方法

- Object.getOwnPropertyDescriptor()：获取某个属性的描述对象。

- Object.defineProperty()：通过描述对象，定义某个属性。
- Object.defineProperties()：通过描述对象，定义多个属性。

3. 控制对象状态的方法

- Object.preventExtensions()：防止对象扩展。
- Object.isExtensible()：判断对象是否可扩展。
- Object.seal()：禁止对象配置。
- Object.isSealed()：判断一个对象是否可配置。
- Object.freeze()：冻结一个对象。
- Object.isFrozen()：判断一个对象是否被冻结。

4. 原型链相关方法

- Object.create()：该方法可以指定原型对象和属性，返回一个新的对象。
- Object.getPrototypeOf()：获取对象的Prototype对象。

4 Object 的实例方法

实例方法是定义在 Object.prototype 原型对象的方法，所有 Object 实例对象都继承了这些方法，通过 `obj.method()` 来调用。

Object实例对象的方法，主要有以下六个：

- Object.prototype.valueOf()：返回当前对象对应的值。
- Object.prototype.toString()：返回当前对象对应的字符串形式。
- Object.prototype.toLocaleString()：返回当前对象对应的本地字符串形式。
- Object.prototype.hasOwnProperty()：判断属性是否为对象自身的属性。
- Object.prototype.isPrototypeOf()：判断当前对象是否为另一个对象的原型。
- Object.prototype.propertyIsEnumerable()：判断某个属性是否可枚举。

---

1. Object.prototype.valueOf()

   返回值： 返回对象原始值(primitive value)。

   ```
   var obj = new Object();
   obj.valueOf() === obj // true
   ```

上面代码比较 obj.valueOf() 与 obj 本身，两者是一样的。

主要用途：自动类型转换

JavaScript 自动类型转换时默认调用这个方法。

```
var obj = new Object();
obj.valueOf = function () {
  return 2;
};

1 + obj // 3
```

上面代码自定义了 obj 对象的 valueOf 方法，于是1 + obj就得到了3。这种方法就相当于用自定义的obj.valueOf，覆盖Object.prototype.valueOf。

2. Object.prototype.toString()

返回值: 返回一个对象的字符串形式。

默认情况下返回类型字符串，形式为：`[object type]`。

```
var o1 = new Object();
o1.toString() // "[object Object]"

var o2 = {a:1};
o2.toString() // "[object Object]"
```

但是通过自定义toString方法，可以让对象在 **自动类型转换** 时，得到想要的字符串形式。

```
var obj = new Object();

obj.toString = function () {
  return 'hello';
};

obj + ' ' + 'world' // "hello world"
```

上面代码表示，当对象用于字符串加法时，会自动调用toString方法。由于自定义了toString方法，所以返回字符串hello world。

自定义 toString 的对象

数组、字符串、函数、Date 对象都分别部署了自定义的toString方法，覆盖了Object.prototype.toString方法。

```
[1, 2, 3].toString() // "1,2,3"

'123'.toString() // "123"

(function () {
```

```
  return 123;
}).toString()
// "function () {
//   return 123;
// }"

(new Date()).toString()
// "Tue May 10 2016 09:11:31 GMT+0800 (CST)"
```

应用：判断数据类型

```
var obj = {};
obj.toString() // "[object Object]"
```

上面代码调用空对象的toString方法，结果返回一个字符串object Object，其中第二个Object表示该对象的类型。

由于实例对象可能会自定义toString方法，覆盖掉Object.prototype.toString方法，所以为了得到类型字符串，最好直接使用Object.prototype.toString方法。通过函数的call方法，可以在任意值上调用这个方法，帮助我们判断这个值的类型。

```
const toString = Object.prototype.toString;

toString.call(2) // "[object Number]"
toString.call('') // "[object String]"
toString.call(true) // "[object Boolean]"
toString.call(undefined) // "[object Undefined]"
toString.call(null) // "[object Null]"
toString.call(Math) // "[object Math]"
toString.call({}) // "[object Object]"
toString.call([]) // "[object Array]"
```

不同数据类型的Object.prototype.toString方法返回值如下。

- 数值：返回`[object Number]`。
- 字符串：返回`[object String]`。
- 布尔值：返回`[object Boolean]`。
- undefined：返回`[object Undefined]`。
- null：返回`[object Null]`。
- 数组：返回`[object Array]`。
- arguments 对象：返回`[object Arguments]`。
- 函数：返回`[object Function]`。
- Error 对象：返回`[object Error]`。

- Date 对象：返回 `[object Date]` 。
- RegExp 对象：返回 `[object RegExp]` 。
- 其他对象：返回 `[object Object]` 。

利用这个特性，可以写出一个比typeof运算符更准确的类型判断函数。

```
var type = function (o){
  var s = Object.prototype.toString.call(o);
  return s.match(/\[object (.*?)\]/)[1].toLowerCase();
};

['Null',
 'Undefined',
 'Object',
 'Array',
 'String',
 'Number',
 'Boolean',
 'Function',
 'RegExp'
].forEach(function (t) {
  type['is' + t] = function (o) {
       return type(o) === t.toLowerCase();
  };
});

type.isObject({}) // true
type.isNumber(NaN) // true
type.isRegExp(/abc/) // true
```

3. Object.prototype.toLocaleString()

返回值： 返回一个对象的本地字符串形式。 这个方法的主要作用是留出一个接口，让各种不同的对象实现自己版本的toLocaleString，用来返回针对某些地域的特定的值。

```
var person = {
  toString: function () {
       return 'Henry Norman Bethune';
  },
  toLocaleString: function () {
       return '白求恩';
  }
};

person.toString() // Henry Norman Bethune
person.toLocaleString() // 白求恩
```

上面代码中，toString()方法返回对象的一般字符串形式，toLocaleString()方法返回本地的字符串形式。

目前，主要有三个对象自定义了toLocaleString方法。

- Array.prototype.toLocaleString()
- Number.prototype.toLocaleString()
- Date.prototype.toLocaleString()

日期的实例对象的toString和toLocaleString返回值就不一样，而且toLocaleString的返回值跟用户设定的所在地域相关。

```
var date = new Date();
date.toString()
// "Sat Sep 19 2020 17:49:53 GMT+0800 (中国标准时间)"

date.toLocaleString()
// "2020/9/19 下午5:50:03"
```

4. Object.prototype.hasOwnProperty()

   传入参数： 字符串。

   返回值： 布尔值。表示该实例对象自身是否具有该属性

```
var obj = {
  p: 123
};

obj.hasOwnProperty('p') // true
obj.hasOwnProperty('toString') // false
```

上面代码中，对象obj自身具有p属性，返回true。但 toString 属性是继承的，返回false。