

深入学习JavaScript对象

JavaScript中，除了五种原始类型（即数字，字符串，布尔值，null，undefined）之外的都是对象了，所以，不把对象学明白怎么继续往下学习呢？

一.概述

对象是一种复合值，它将很多值（原始值或其他对象）聚合在一起，可通过属性名访问这些值。而属性名可以是包含空字符串在内的任意字符串。JavaScript对象也可以称作一种数据结构，正如我们经常听说的“散列（hash）”、“散列表（hashtable）”、“字典（dictionary）”、“关联数组（associative array）”。

JavaScript中对象可以分为三类：

- ①内置对象，例如数组、函数、日期等；
- ②宿主对象，即JavaScript解释器所嵌入的宿主环境（比如浏览器）定义的，例如HTMLElement等；
- ③自定义对象，即程序员用代码定义的；

对象的属性可以分为两类：

- ①自有属性（own property）：直接在对象中定义的属性；
- ②继承属性（inherited property）：在对象的原型对象中定义的属性（关于原型对象下面会详谈）；

二.对象的创建

既然学习对象，又怎能不懂如何创建对象呢？面试前端岗位的同学，可能都被问过这个基础问题吧：

创建JavaScript对象的两种方法是什么？（或者：说说创建JavaScript对象的方法？）

这个问题我就被问过两次。“创建对象的两种方法”这种说法网上有很多，但是据我所看书籍来说是有三种方法的！下面我们就来具体谈谈这三种方法：

1.对象直接量

对象直接量由若干名/值对组成的映射表，名/值对中间用冒号分隔，名/值对之间用逗号分隔，整个映射表用花括号括起来。属性名可以是JavaScript标识符也可以是字符串直接量，也就是说下面两种创建对象obj的写法是完全一样的：

```
var obj = {x: 1, y: 2};
var obj = {'x': 1, 'y':2};
```

2.通过new创建对象

new运算符后跟随一个函数调用，即构造函数，创建并初始化一个新对象。例如：

```
1 var o = new Object();    //创建一个空对象，和{}一样
2 var a = new Array();     //创建一个空数组，和[]一样
3 var d = new Date();      //创建一个表示当前时间的Date对象
```

关于构造函数相关的内容以后再说。

公告

昵称：clearbug
园龄：3年7个月
粉丝：42
关注：15
[+加关注](#)

< 2018年12			
日	一	二	三
25	26	27	28
2	3	4	5
9	10	11	12
16	17	18	19
23	24	25	26
30	31	1	2

随笔分类

ActiveMQ(16)

cmd(2)

CSS(1)

C语言(1)

Git(1)

Java 文档(2)

Java9(1)

JavaScript(16)

JVM(1)

MacOS(1)

Shiro(1)

3.Object.create()

ECMAScript5定义了一个名为Object.create()的方法，它创建一个新对象，其中第一个参数是这个对象的原型对象（好像还没解释原型对象...下面马上就说），第二个可选参数用以对对象的属性进行进一步的描述，第二个参数下面再说（因为这第三种方法是ECMAScript5中定义的，所以以前大家才经常说创建对象的两种方法的吧？个人觉得应该是这个原因）。这个方法使用很简单：

```
1 var o1 = Object.create({x: 1, y: 2});    //对象o1继承了属性x和y
2 var o2 = Object.create(null);           //对象o2没有原型
```

下面三种的完全一样的：

```
1 var obj1 = {};
2 var obj2 = new Object();
3 var obj3 = Object.create(Object.prototype);
```

为了解释为啥这三种方式是完全一样的，我们先来解释下JavaScript中的原型对象（哎，让客官久等了！），记得一位大神说过：

Javascript是一种基于对象（object-based）的语言，你遇到的所有东西几乎都是对象。但是，它又不是一种真正的面向对象编程（OOP）语言，因为它的语法中没有class（类）。

面向对象的编程语言JavaScript，没有类!!!那么，它是如何实现继承的呢？没错，就是通过原型对象。基本上每一个JavaScript对象（null除外）都和另一个对象相关联，“另一个”对象就是所谓的原型对象（原型对象也可以简称为原型，并没有想象的那么复杂，它也只是是一个对象而已）。每一个对象都从原型对象继承属性，并且一个对象的prototype属性的值（这个属性在对象创建时默认自动生成，并不需要显示的自定义）就是这个对象的原型对象，即obj.prototype就是对象obj的原型对象。

原型对象先说到这，回到上面的问题，有了对原型对象的认识，下面就是不需要过多解释的JavaScript语言规定了：

- ①所有通过对象直接量创建的对象的原型对象就是Object.prototype对象；
- ②通过关键字new和构造函数创建的对象的原型对象就是构造函数prototype属性的值，所以通过构造函数Object创建的对象的原型就是Object.prototype了；

现在也补充了第三种创建对象的方法Object.create()第一个参数的含义。

三.属性的查询和设置

学会了如何创建对象还不够啊，因为对象只有拥有一些属性才能真正起到作用滴！那么，就继续往下学习对象的属性吧！

可以通过点（.）或方括号（[]）运算符来获取和设置属性的值。对于点（.）来说，右侧必须是一个以属性名命名的标识符（注意：JavaScript语言的标识符有自己的合法规则，并不同于带引号的字符串）；对于方括号（[]）来说，方括号内必须是一个字符串表达式（字符串变量当然也可以喽，其他可以转换成字符串的值比如数字什么的也是都可以滴），这个字符串就是属性的名字。正如下面例子：

```
1 var obj = {x: 1, y: 2};
2 obj.x = 5;
3 obj['y'] = 6
```

概述中说过，JavaScript对象具有“自有属性”，也有“继承属性”。当查询对象obj的属性x时，首先会查找对象obj自有属性中是否有x，如果没有，就会查找对象obj的原型对象obj.prototype是否有属性x，如果没有，就会进而查找对象obj.prototype的原型对象obj.prototype.prototype是否有属性x，就这样直到找到x或者查找到的原型对象是undefined的对象为止。可以看到，一个对象上面继承了很多原型对象，这些原型对象就构成了一个“链”，这也就是我们平时所说的“原型链”，这种继承也就是JavaScript中“原型式继承”（prototypal inheritance）。

对象o查询某一属性时正如上面所说会沿着原型链一步步查找，但是其设置某一属性的值时，只会修改自有属性（如果对象没有这个属性，那就会添加这个属性并赋值），并不会修改原型链上其他对象的属性。

四.存取器属性getter和setter

上面我们所说的都是很普通的对象属性，这种属性称做“数据属性”（data property），数据属性只有一个简单的值。然而在ECMAScript 5中，属性值可以用一个或两个方法替代，这两个方法就是getter和setter，有getter和setter定义的属性称做“存取器属性”（accessor property）。

Think in Java(8)
Ubuntu(2)
命令流
深入浅出 MySQL(6)
书单(1)
网络协议(1)
文学(2)

阅读排行榜
1. JavaScript之对象序7)
2. JavaScript中数字与(21515)
3. 深入学习JavaScript
4. 初识 JShell(3873)
5. JavaScript之事件处


评论排行榜
1. JavaScript之数组去
2. 深入学习JavaScript
3. JavaScript之函数作
4. JavaScript之事件处
5. 关于JavaScript中的

推荐排行榜
1. 深入学习JavaScript
2. JavaScript之对象序
3. JavaScript之函数作
4. JavaScript之事件处
5. JavaScript之数组去

当程序查询存取器属性的值时，JavaScript调用getter方法（无参数）。这个方法的返回值就是属性存取表达式的值。当程序设置一个存取器属性的值时，JavaScript调用setter方法，将赋值表达式右侧的值当做参数传入setter。如果属性同时具有getter和setter方法，那么它就是一个读/写属性；如果它只有getter方法，那么它就是一个只读属性，给只读属性赋值不会报错，但是并不能成功；如果它只有setter方法，那么它是一个只写属性，读取只写属性总是返回undefined。看个实际的例子：

```
1 var p = {
2   x: 1.0,
3   y: 2.0,
4   get r(){ return Math.sqrt(this.x*this.x + this.y*this.y); },
5   set r(newvalue){
6     var oldvalue = Math.sqrt(this.x*this.x + this.y*this.y);
7     var ratio = newvalue/oldvalue;
8     this.x *= ratio;
9     this.y *= ratio;
10  },
11  get theta(){ return Math.atan2(this.y, this.x); },
12  print: function(){ console.log('x:'+this.x+', y:'+this.y); }
13};
```

正如例子所写，存取器属性定义一个或两个和属性同名的函数，这个函数定义并没有使用function关键字，而是使用get和set，也没有使用冒号将属性名和函数体分隔开。对比一下，下面的print属性是一个函数方法。注意：这里的getter和setter里this关键字的用法，JavaScript把这些函数当做对象的方法来调用，也就是说，在函数体内的this指向这个对象。下面看下实例运行结果：



```
JavaScript
var p = {
  x: 1.0,
  y: 2.0,
  get r(){ return Math.sqrt(this.x*this.x + this.y*this.y); },
  set r(newvalue){
    var oldvalue = Math.sqrt(this.x*this.x + this.y*this.y);
    var ratio = newvalue/oldvalue;
    this.x *= ratio;
    this.y *= ratio;
  },
  get theta(){ return Math.atan2(this.y, this.x); },
  print: function(){ console.log('x:'+this.x+', y:'+this.y); }
};
console.log(p);
```

```
Console
[Object Object] {
  print: function () {
    window.runnerWindow.proxyConsole.log('x:'+this.x+', y:'+this.y); },
  r: 2.23606797749979,
  theta: 1.1071487177940904,
  x: 1,
  y: 2
}
```

正如控制台的输出，r、theta同x、y一样只是一个值属性，print是一个方法属性。

ECMAScript 5增加的这种存取器，虽然比普通属性更为复杂了，但是也使得操作对象属性键值对更加严谨了。

五.删除属性

程序猿撸码一般都是实现增、删、改、查功能，前面已经说了增、改、查，下面就说说删除吧！

delete运算符可以删除对象的属性，它的操作数应该是一个属性访问表达式。但是，**delete只是断开属性和宿主对象的联系，而不会去操作属性中的属性：**

```
1 var a = {p:{x:1}};
2 var b = a.p;
3 delete a.p;
```

执行这段代码后b.x的值依然是1，由于已删除属性的引用依然存在，所以有时这种不严谨的代码会造成内存泄露，所以在销毁对象的时候，要遍历属性中的属性，依次删除。

delete表达式返回true的情况：

- ①删除成功或没有任何副作用（比如删除不存在的属性）时；
- ②如果delete后不是一个属性访问表达式；

```
1 var obj = {x: 1, get r(){return 5;}, set r(newvalue){this.x = newvalue;}};
2 delete obj.x; //删除对象obj的属性x, 返回true
3 delete obj.x; //删除不存在的属性, 返回true
4 delete obj.r; //删除对象obj的属性r, 返回true
```

```
5 delete obj.toString; //没有任何副作用 (toString是继承来的, 并不能删除), 返回true
6 delete 1; //数字1不是属性访问表达式, 返回true
```



delete表达式返回false的情况:

①删除可配置性(可配置性是属性的一种特性, 下面会谈)为false的属性时;



```
1 delete Object.prototype; //返回false, prototype属性是不可配置的
2 //通过var声明的变量或function声明的函数是全局对象的不可配置属性
3 var x = 1;
4 delete this.x; //返回false
5 function f() {}
6 delete this.f; //返回false
```



六.属性的特性

上面已经说到了属性的可配置性特性, 因为下面要说的检测属性和枚举属性还要用到属性的特性这些概念, 所以现在就先具体说说属性的特性吧!

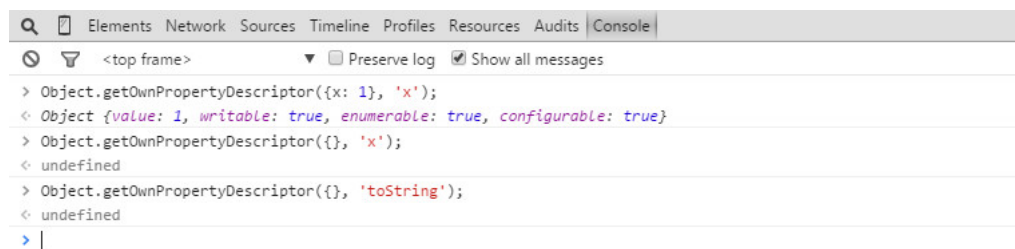
除了包含名字和值之外, 属性还包含一些标识它们可写、可枚举、可配置的三种特性。在ECMAScript 3中无法设置这些特性, 所有通过ECMAScript 3的程序创建的属性都是可写的、可枚举的和可配置的, 且无法对这些特性做修改。ECMAScript 5中提供了查询和设置这些属性特性的API。这些API对于库的开发者非常有用, 因为:

①可以通过这些API给原型对象添加方法, 并将它们设置成不可枚举的, 这让它们更像内置方法;

②可以通过这些API给对象定义不能修改或删除的属性, 借此“锁定”这个对象;

在这里我们将存取器属性的getter和setter方法看成是属性的特性。按照这个逻辑, 我们也可以把属性的值同样看做属性的特性。因此, 可以认为属性包含一个名字和4个特性。数据属性的4个特性分别是它的值 (value)、可写性 (writable)、可枚举性 (enumerable) 和可配置性 (configurable)。存取器属性不具有值特性和可写性它们的可写性是由setter方法是否存在与否决定的。因此存取器属性的4个特性是读取 (get)、写入 (set)、可枚举性和可配置性。

为了实现属性特性的查询和设置操作, ECMAScript 5中定义了一个名为“属性描述符” (property descriptor) 的对象, 这个对象代表那4个特性。描述符对象的属性和它们所描述的属性特性是同名的。因此, 数据属性的描述符对象的属性有value、writable、enumerable和configurable。存取器属性的描述符对象则用get属性和set属性代替value和writable。其中writable、enumerable和configurable都是布尔值, 当然, get属性和set属性是函数值。通过调用Object.getOwnPropertyDescriptor()可以获得某个对象特定属性的属性描述符:



```
> Object.getOwnPropertyDescriptor({x: 1}, 'x');
< Object {value: 1, writable: true, enumerable: true, configurable: true}
> Object.getOwnPropertyDescriptor({}, 'x');
< undefined
> Object.getOwnPropertyDescriptor({}, 'toString');
< undefined
> |
```

从函数名字就可以看出, Object.getOwnPropertyDescriptor()只能得到自有属性的描述符, 对于继承属性和不存在的属性它都返回undefined。要想获得继承属性的特性, 需要遍历原型链 (不会遍历原型链? 不要急, 下面会说到的)。

要想设置属性的特性, 或者想让新建属性具有某种特性, 则需要调用Object.defineProperty(), 传入需要修改的对象、要创建或修改的属性的名称以及属性描述符对象。

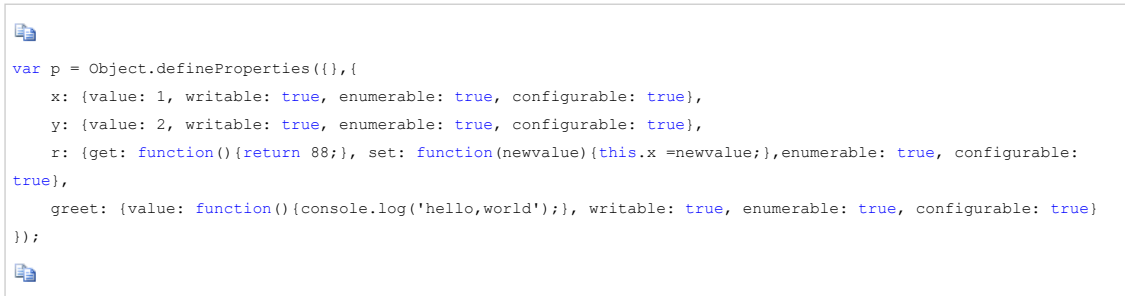


```
> var obj = {};  
< undefined  
> console.log(obj);  
Object {}  
< undefined  
> Object.defineProperty(obj, 'x', {value: 1, writable: true, enumerable:false, configurable: true});  
< Object {x: 1}  
> obj.x = 2;  
< 2  
> obj  
< Object {x: 2}  
> Object.defineProperty(obj, 'x', {writable: false});  
< Object {x: 2}  
> obj.x = 3;  
< 3  
> obj  
< Object {x: 2}  
> Object.defineProperty(obj, 'x', {value: 5});  
< Object {x: 5}  
  
> Object.defineProperty(obj, 'y', {get: function(){return 88;}});  
< ▼ Object {x: 5} ⓘ  
  x: 5  
  y: (...)  
    ▶ get y: function ()  
    ▶ __proto__: Object  
> obj.y  
< 88
```

可以看到：

- ①传入Object.defineProperty()的属性描述符对象不必包含所有4个特性；
- ②可写性控制着对属性值的修改；
- ③可枚举性控制着属性是否可枚举（枚举属性，下面会说的）；
- ④可配置性控制着对其他特性（包括前面说过的属性是否可以删除）的修改；

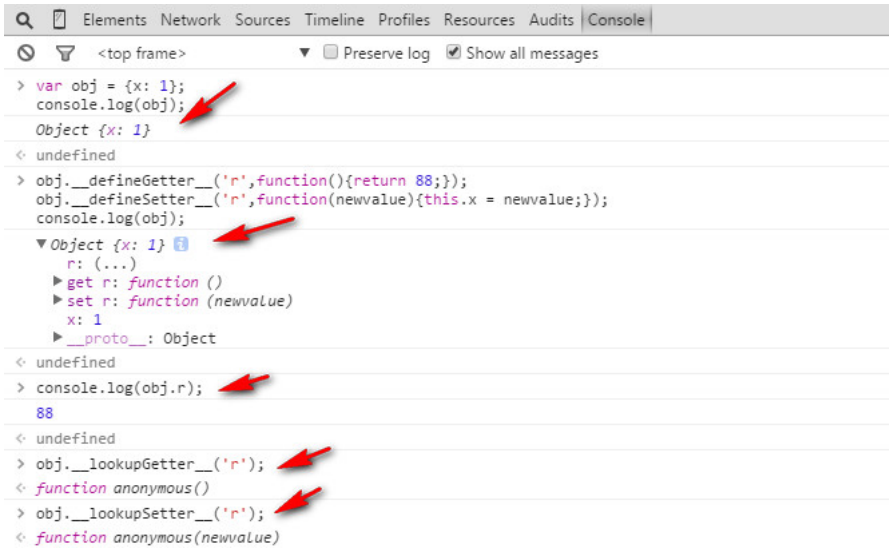
如果要同时修改或创建多个属性，则需要使用Object.defineProperties()。第一个参数是要修改的对象，第二个参数是一个映射表，它包含要新建或修改的属性的名称，以及它们的属性描述符，例如：



```
var p = Object.defineProperties({}, {  
  x: {value: 1, writable: true, enumerable: true, configurable: true},  
  y: {value: 2, writable: true, enumerable: true, configurable: true},  
  r: {get: function(){return 88;}, set: function(newvalue){this.x =newvalue;}, enumerable: true, configurable: true},  
  greet: {value: function(){console.log('hello,world');}, writable: true, enumerable: true, configurable: true}  
});
```

相信你也已经从实例中看出：Object.defineProperty()和Object.defineProperties()都返回修改后的对象。

前面我们说getter和setter存取器属性时使用对象直接量语法给新对象定义存取器属性，但并不能查询属性的getter和setter方法或给已有的对象添加新的存取器属性。在ECMAScript 5中，就可以通过Object.getOwnPropertyDescriptor()和Object.defineProperty()来完成这些工作啦！但在ECMAScript 5之前，大多数浏览器（IE除外啦）已经支持对象直接量语法中的get和set写法了。所以这些浏览器还提供了非标准的老式API用来查询和设置getter和setter。这些API有4个方法组成，所有对象都拥有这些方法。lookupGetter ()和 lookupSetter ()用以返回一个命名属性的getter和setter方法。__defineGetter__ ()和 __defineSetter__ ()用以定义getter和setter。这四个方法都是以两条下划线做前缀，两条下划线做后缀，以表明它们是非标准方法。下面是它们用法：



```
var obj = {x: 1};
console.log(obj);
Object {x: 1}

obj.__defineGetter__('r',function(){return 88;});
obj.__defineSetter__('r',function(newvalue){this.x = newvalue;});
console.log(obj);
Object {x: 1}
  r: (...)
    get r: function ()
    set r: function (newvalue)
    x: 1
    __proto__: Object

console.log(obj.r);
88

obj.__lookupGetter__('r');
function anonymous()

obj.__lookupSetter__('r');
function anonymous(newvalue)
```

七.检测属性

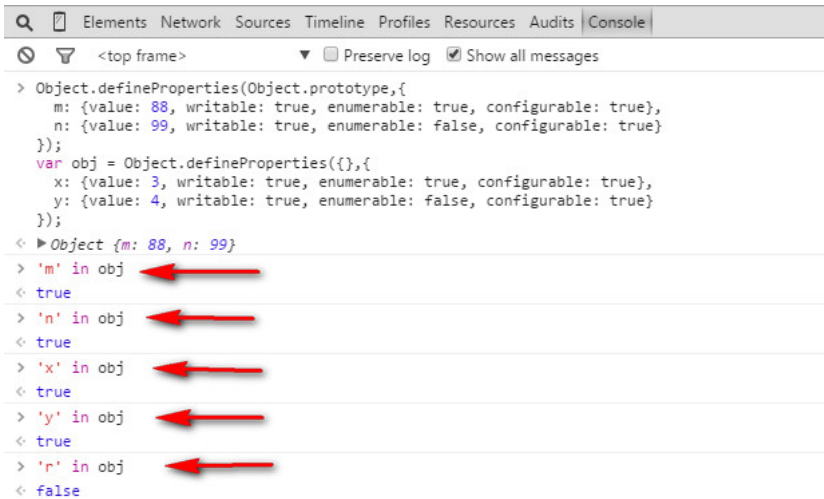
JavaScript对象可以看做属性的集合，那么我们就需要判断某个属性是否存在于某个对象中，这就是接下来要说的检测属性。

检测一个对象的属性也有三种方法，下面就来详细说说它们的作用及区别！

1.in运算符

in运算符左侧是属性名（字符串），右侧是对象。如果对象的自有属性或继承属性中包含这个属性则返回true，否则返回false。

为了试验，我们先给对象Object.prototype添加一个可枚举属性m，一个不可枚举属性n；然后，给对象obj定义两个可枚举属性x，一个不可枚举属性y，并且对象obj是通过对象直接量形式创建的，继承了Object.prototype。下面看实例：



```
Object.defineProperties(Object.prototype,{
  m: {value: 88, writable: true, enumerable: true, configurable: true},
  n: {value: 99, writable: true, enumerable: false, configurable: true}
});
var obj = Object.defineProperties({},{
  x: {value: 3, writable: true, enumerable: true, configurable: true},
  y: {value: 4, writable: true, enumerable: false, configurable: true}
});
Object {m: 88, n: 99}

'm' in obj
true

'n' in obj
true

'x' in obj
true

'y' in obj
true

'r' in obj
false
```

从运行结果可以看出：in运算符左侧是属性名（字符串），右侧是对象。如果对象的自有属性或继承属性（不论这些属性是否可枚举）中包含这个属性则返回true，否则返回false。

2.hasOwnProperty()

对象的hasOwnProperty()方法用来检测给定的名字是否是对象的自有属性（不论这些属性是否可枚举），对于继承属性它将返回false。下面看实例：


```
Q Elements Network Sources Timeline Profiles Resources Audits Console
< top frame > Preserve log Show all messages

> Object.defineProperty(Object.prototype,{
  m: {value: 88, writable: true, enumerable: true, configurable: true},
  n: {value: 99, writable: true, enumerable: false, configurable: true}
});
var obj = Object.defineProperty({},{
  x: {value: 3, writable: true, enumerable: true, configurable: true},
  y: {value: 4, writable: true, enumerable: false, configurable: true}
});
< ▶ Object {m: 88, n: 99}
> obj.hasOwnProperty('m');
< false
> obj.hasOwnProperty('n');
< false
> obj.hasOwnProperty('x');
< true
> obj.hasOwnProperty('y');
< true
```

3.propertyIsEnumerable()

propertyIsEnumerable()是hasOwnProperty()的增强版，只有检测到是自有属性且这个属性可枚举性为true时它才返回true。还是实例：

```
Q Elements Network Sources Timeline Profiles Resources Audits Console
< top frame > Preserve log Show all messages

> Object.defineProperty(Object.prototype,{
  m: {value: 88, writable: true, enumerable: true, configurable: true},
  n: {value: 99, writable: true, enumerable: false, configurable: true}
});
var obj = Object.defineProperty({},{
  x: {value: 3, writable: true, enumerable: true, configurable: true},
  y: {value: 4, writable: true, enumerable: false, configurable: true}
});
< ▶ Object {m: 88, n: 99}
> obj.propertyIsEnumerable('m');
< false
> obj.propertyIsEnumerable('n');
< false
> obj.propertyIsEnumerable('x');
< true
> obj.propertyIsEnumerable('y');
< false
```

八.枚举属性

相对于检测属性，我们更常用的是枚举属性。枚举属性我们通常使用for/in循环，它可以在循环体中遍历对象中所有可枚举的自有属性和继承属性，把属性名称赋值给循环变量。继续上实例：

```
Q Elements Network Sources Timeline Profiles Resources Audits Console
< top frame > Preserve log Show all messages

> Object.defineProperty(Object.prototype,{
  m: {value: 88, writable: true, enumerable: true, configurable: true},
  n: {value: 99, writable: true, enumerable: false, configurable: true}
});
var obj = Object.defineProperty({},{
  x: {value: 3, writable: true, enumerable: true, configurable: true},
  y: {value: 4, writable: true, enumerable: false, configurable: true}
});
< ▶ Object {m: 88, n: 99}
> for(prop in obj){
  console.log(prop);
}
x
m
< undefined
> |
```

我原来认为for/in循环跟in运算符有莫大关系的，现在看来它们的规则并不相同啊！当然，如果这里不想遍历出继承的属性，那就在for/in循环中加一层hasOwnProperty()判断：

```
for(prop in obj){
  if(obj.hasOwnProperty(prop)){
    console.log(prop);
  }
}
```

除了for/in循环之外，ECMAScript 5还定义了两个可以枚举属性名称的函数：

①Object.getOwnpropertyNames()，它返回对象的所有自有属性的名称，不论是否可枚举；

②Object.keys(), 它返回对象对象中可枚举的自有属性的名称;

还是实例:

```
Object.defineProperty(Object.prototype, {
  m: {value: 88, writable: true, enumerable: true, configurable: true},
  n: {value: 99, writable: true, enumerable: false, configurable: true}
});
var obj = Object.defineProperty({}, {
  x: {value: 3, writable: true, enumerable: true, configurable: true},
  y: {value: 4, writable: true, enumerable: false, configurable: true}
});
Object {m: 88, n: 99}
Object.getOwnPropertyNames(obj);
["x", "y"]
Object.keys(obj);
["x"]
```

九.对象的三个特殊属性

每个对象都有与之相关的原型 (prototype)、类 (class) 和可扩展性 (extensible attribute)。这三个就是对象的特殊属性 (它们也只是对象的属性而已, 并没有想象的复杂哦)。

1.原型属性

正如前面所说, 对象的原型属性是用来继承属性的 (有点绕...), 这个属性如此重要, 以至于我们经常把“o的原型属性”直接叫做“o的原型”。原型属性是在实例创建之初就设置好的 (也就是说, 这个属性的值是JavaScript默认自动设置的, 后面我们会说如何自己手动设置), 前面也提到:

①通过对象直接量创建的对象使用Object.prototype作为它们的原型;

②通过new+构造函数创建的对象使用构造函数的prototype属性作为它们的原型;

③通过Object.create()创建的对象使用第一个参数 (如果这个参数为null, 则对象原型属性值为undefined; 如果这个参数为undefined, 则会报错: Uncaught TypeError: Object prototype may only be an Object or null: undefined) 作为它们的原型;

那么, 如何查询一个对象的原型属性呢? 在ECMAScript 5中, 将对象作为参数传入Object.getPrototypeOf()可以查询它的原型, 例如:

```
var obj = Object.create(Object.prototype);
Object {}
Object.getPrototypeOf(obj);
Object {}
  __defineGetter__: function __defineGetter__()
  __defineSetter__: function __defineSetter__()
  __lookupGetter__: function __lookupGetter__()
  __lookupSetter__: function __lookupSetter__()
  constructor: function Object()
  hasOwnProperty: function hasOwnProperty()
  isPrototypeOf: function isPrototypeOf()
  propertyIsEnumerable: function propertyIsEnumerable()
  toLocaleString: function toLocaleString()
  toString: function toString()
  valueOf: function valueOf()
  get __proto__: function get __proto__()
  set __proto__: function set __proto__()
Object.getPrototypeOf(obj) === Object.prototype
true
```

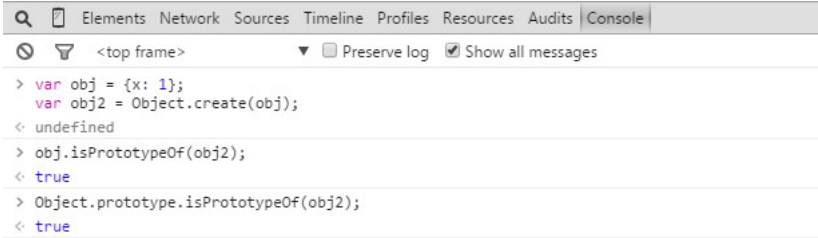
但是在ECMAScript 3中, 没有Object.getPrototypeOf()函数, 但经常使用表达式obj.constructor.prototype来检测一个对象的原型, 因为每个对象都有一个constructor属性表示这个对象的构造函数:

①通过对象直接量创建的对象.constructor属性指向构造函数Object();

②通过new+构造函数创建的对象.constructor属性指向构造函数;

③通过Object.create()创建的对象.constructor属性指向与其原型对象的constructor属性指向相同;

要检测一个对象是否是另一个对象的原型 (或处于原型链中), 可以使用isPrototypeOf()方法。例如:

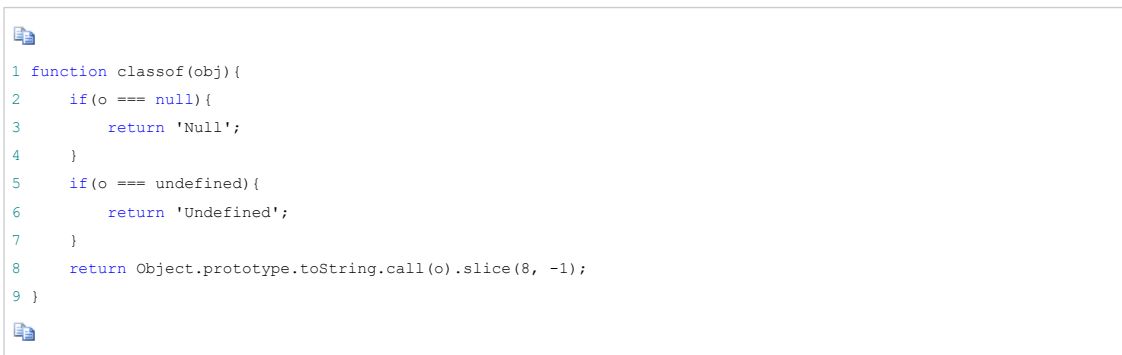


```
Q Elements Network Sources Timeline Profiles Resources Audits Console
<top frame> Preserve log Show all messages
> var obj = {x: 1};
var obj2 = Object.create(obj);
< undefined
> obj.isPrototypeOf(obj2);
< true
> Object.prototype.isPrototypeOf(obj2);
< true
```

还有一个非标准但众多浏览器都已实现的对象的属性`__proto__`(同样是两个下划线开始和结束, 以表明其为非标准), 用以直接查询/设置对象的原型。

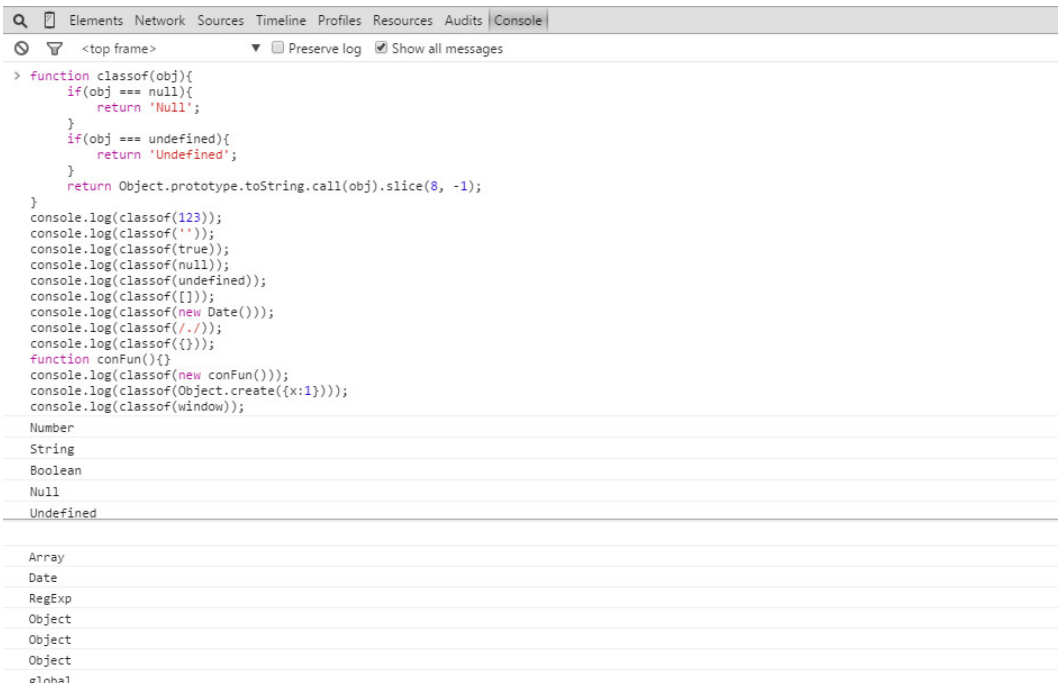
2. 类属性

对象的类属性 (class attribute) 是一个字符串, 用以表示对象的类型信息。ECMAScript 3 和ECMAScript 5 都未提供设置这个属性的方法, 并只有一种间接的方法可以查询它。默认的`toString()`方法 (继承自`Object.prototype`) 返回了这种格式的字符串: `[object class]`。因此, 要想获得对象的类, 可以调用对象的`toString()`方法, 然后提取已返回字符串的第8到倒数第二个位置之间的字符。不过, 很多对象继承的`toString()`方法重写了 (比如: `Array`、`Date`等), 为了能调用正确的`toString()`版本, 必须间接地调用`Function.call()`方法。下面代码可以返回传递给它的任意对象的类:



```
1 function classof(obj){
2     if(o === null){
3         return 'Null';
4     }
5     if(o === undefined){
6         return 'Undefined';
7     }
8     return Object.prototype.toString.call(o).slice(8, -1);
9 }
```

`classof()`函数可以传入任何类型的参数。下面是使用实例:



```
Q Elements Network Sources Timeline Profiles Resources Audits Console
<top frame> Preserve log Show all messages
> function classof(obj){
  if(obj === null){
    return 'Null';
  }
  if(obj === undefined){
    return 'Undefined';
  }
  return Object.prototype.toString.call(obj).slice(8, -1);
}
console.log(classof(123));
console.log(classof(''));
console.log(classof(true));
console.log(classof(null));
console.log(classof(undefined));
console.log(classof([]));
console.log(classof(new Date()));
console.log(classof(/. /));
console.log(classof({}));
function conFun(){
  console.log(classof(new conFun()));
  console.log(classof(Object.create({x:1})));
  console.log(classof(window));
}
Number
String
Boolean
Null
Undefined
Array
Date
RegExp
Object
Object
Object
Object
global
```

总结: 从运行结果可以看出通过三种方式创建的对象类属性都是'Object'。

3. 可扩展性

对象的可扩展性用以表示是否可以给对象添加新属性。所有内置对象和自定义对象都是显示可扩展的 (除非将它们转换为不可扩展), 宿主对象的可扩展性是由JavaScript引擎定义的。ECMAScript 5中定义了用来查询和设置对象可扩展性的函数:

- ① (查询) 通过将对象传入`Object.isExtensible()`, 来判断该对象是否是可扩展的。

②（设置）如果想将对象转换为不可扩展，需要调用`Object.preventExtensions()`，将待转换的对象作为参数传进去。注意：

a.一旦将对象转换为不可扩展的，就无法再将其转换回可扩展的了；

b.`preventExtensions()`只影响到对象本身的可扩展性，如果给一个不可扩展的对象的原型添加属性，这个不可扩展的对象同样会继承这些新属性；

进一步，`Object.seal()`和`Object.preventExtensions()`类似，除了能将对象设置为不可扩展的，还可以将对象的所有自有属性都设置为不可配置的。对于那些已经封闭（sealed）起来的对象是不能解封的。可以使用`Object.isSealed()`来检测对象是否封闭。

更进一步，`Object.freeze()`将更严格地锁定对象——“冻结”（frozen）。除了将对象设置为不可扩展和将其属性设置为不可配置之外，还可以将它自有的所有数据属性设置为只读（若对象的存取器属性有setter方法，存取器属性将不受影响，仍可通过给属性赋值调用它们）。使用`Object.isFrozen()`来检测对象是否冻结。

总结：`Object.preventExtensions()`、`Object.seal()`和`Object.freeze()`都返回传入的对象，也就是说，可以通过嵌套的方式调用它们：

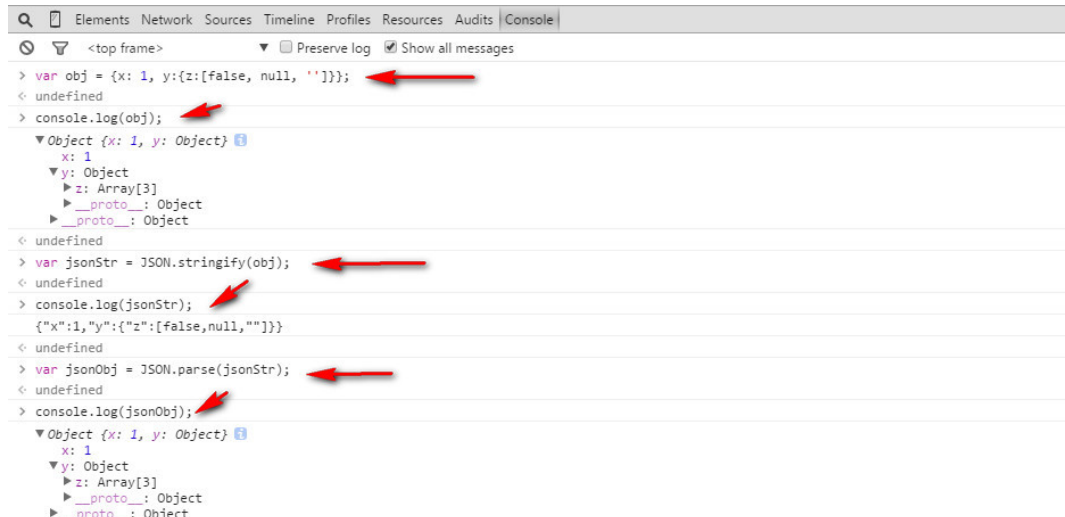
```
1 var obj = Object.seal(Object.create(Object.freeze({x:1}),{y:{value: 2, writable: true}}));
```

这条语句中使用`Object.create()`函数传入了两个参数，即第一个参数是创建出的对象的原型对象，第二个参数是在创建对象是直接给其定义的属性，并且附带定义了属性的特性。

十.对象的序列化

前面说完了对象的属性以及对象属性的特性，东西还是蛮多的，不知道你是否已看晕。不过，下面就是比较轻松的话题了！

对象序列化（serialization）是指将对象的状态转换为字符串，也可以将字符串还原为对象。ECMAScript 5提供了内置函数`JSON.stringify()`和`JSON.parse()`用来序列化和还原对象。这些方法都使用JSON作为数据交换格式，JSON的全称是“JavaScript Object Notation”——JavaScript对象表示法，它的语法和JavaScript对象与数组直接量的语法非常相近：



其中，最后的`jsonObj`是`obj`的深拷贝（关于什么是深拷贝，什么是浅拷贝，可以参考：<http://www.zhihu.com/question/23031215>，第二个答案）。

JSON的语法是JavaScript的子集，它并不能表示JavaScript里的所有值。支持对象、数组、字符串、无穷大数字、`true`、`false`和`null`，并且它们可以序列化和还原。注意：

①`NaN`、`Infinity`和`-Infinity`序列化的结果是`null`；

②`JSON.stringify()`只能序列化对象可枚举的自有属性；

③日期对象序列化的结果是ISO格式的日期字符串（参照`Date.toJSON()`函数），但`JSON.parse()`依然保留它们的字符串形态，而不能将它们还原为原始日期对象；

④函数、RegExp、Error对象和undefined值不能序列化和还原；

当然，JSON.stringify()和JSON.parse()都可以接受第二个可选参数，通过传入需要序列化或还原的属性列表来定制自定义的序列化或还原操作，这个我们以后再详谈。

下面是在下衷心送给各位看官的五个欢迎 (O(∩_∩)O~)：欢迎转载、欢迎收藏、欢迎评论、欢迎点赞、欢迎推荐！

分类： JavaScript

标签： JavaScript, 对象

好文要顶

关注我

收藏该文

clearbug

关注 - 15

粉丝 - 42

+加关注

240

« 上一篇：关于JavaScript中的事件代理
» 下一篇：JavaScript学习之获取URL参数

posted @ 2015-09-28 08:53 clearbug 阅读(4236) 评论(23) 编辑 收藏

评论列表

1楼 2015-09-28 09:13 论谁是英雄

辛苦！

支持(0) 反对(0)

2楼[楼主] 2015-09-28 09:22 clearbug

@ 论谁是英雄

恩，谢谢，以前只是简单的使用对象，今天才发现JavaScript中对对象有那么多可以学习的东西

支持(0) 反对(0)

3楼 2015-09-28 09:49 m-cat

都是干货，继续支持

支持(0) 反对(0)

4楼 2015-09-28 09:51 懒得安分

楼主有时间可以写写JavaScript通过Prototype继承的相关知识点。。。

支持(0) 反对(0)

5楼[楼主] 2015-09-28 09:52 clearbug

@ 猫神战之京

谢谢支持，让我有继续写的动力！

支持(0) 反对(0)

6楼[楼主] 2015-09-28 09:53 clearbug

@ 懒得安分

嗯。正准备下次写关于构造函数和继承的知识呢

支持(0) 反对(0)

7楼 2015-09-28 11:00 Jokerone

赞，写得不错！！

支持(0) 反对(0)

8楼[楼主] 2015-09-28 11:02 clearbug

@ Jokerone
谢谢!

支持(0) 反对(0)

#9楼 2015-09-28 13:32 请叫我头头哥

不错。赞一个

支持(0) 反对(0)

#10楼[楼主] 2015-09-28 13:44 clearbug

@ 请叫我头头哥
谢赞!

支持(0) 反对(0)

#11楼 2015-09-28 21:53 落叶飞逝的恋

界面不错，楼主能分享下css吗?

支持(0) 反对(0)

#12楼[楼主] 2015-09-28 22:55 clearbug

@ 落叶飞逝的恋
我直接用自带的皮肤得——CodingLife

支持(0) 反对(0)

#13楼 2015-09-29 00:14 落叶飞逝的恋

@ clearbug
蛮漂亮的谢谢

支持(0) 反对(0)

#14楼 2015-09-29 08:12 天翔e派

不错，辛苦了，已收藏

支持(0) 反对(0)

#15楼[楼主] 2015-09-29 08:52 clearbug

@ 天翔e派
谢谢支持!

支持(0) 反对(0)

#16楼 2015-12-29 11:03 念念伊梦

楼主，在下认为你有个地方表述不恰当
对象的属性可以分为两类：
①自有属性（own property）：直接在对象中定义的属性；
②继承属性（inherited property）：在对象的原型对象中定义的属性（关于原型对象下面会详谈）；

根据你想表达的意思，这里的分类维度定义，不应该是属性，而是 成员，因为相对于对象来说 属性是定义对象的客观描述，方法是对对象的行为表述
所以因为描述为，
对象的成员可分为，
实例成员：（就是你说的自有属性） 实例成员又包括 ， 实例属性 ， 实例方法
原型成员：（就是你说的继承属性） 原型成员又包括 ， 原型属性 ， 原型方法

就是说，在JavaScript中， 属性和方法是完全不同的两个定义，不能混而论之，你把 属性和方法 统称为了属性，把属性叫做值属性，方法叫走方法属性，这会给人一些错误的引导的。

支持(0) 反对(0)

#17楼[楼主] 2015-12-29 11:18 clearbug

@ 念念伊梦

额。我看你说的问题主要就是在“属性”这一次的含义上面，看你说的“属性”的意思就是一些值，而“方法”的意思就是一些对象的函数，个人感觉这么说的话放在Java里很合适。不过，我感觉javascript这种弱类型脚本语言中，类型界定不是那么明显；一个对象里的函数，是一个方法，也是一个对象哇。上面只是个人的一点浅显见解，其实我当时这么写的主要原因还是因为看的《javascript权威指南》上面这么写的。

支持(0) 反对(0)

#18楼 2015-12-29 11:26 念念伊梦

@ clearbug

嗯，JavaScript是弱类型的，不过就对象的概念而言，它和所有遵从OOP的语言语言一样，万物皆对象，只不过funciton类型是特殊的对象，也是JavaScript里面的一级对象。

关于属性和方法的定义，之所以区分开，是因为他们本身的作用不同，你获取一个属性通过. 或者 [] 可以得到，但是要调用方法，是需要通过() 来执行的。

嗯，关于对象的基础知识你写的还是比较详细的，主要是，如果对基础知识不熟悉的人，这里的属性，方法，值属性，方法属性，会给人绕晕的。

支持(0) 反对(0)

#19楼[楼主] 2015-12-29 11:29 clearbug

@ 念念伊梦

嗯。要谢谢你的评论哈。以后有人如果因为这里绕晕的话，看到你的评论也能明了了！

支持(0) 反对(0)

#20楼 2015-12-29 11:31 念念伊梦

@ clearbug

我也是才疏学浅，个人见解，楼主见笑了。O(∩_∩)O哈哈~

支持(0) 反对(0)

#21楼[楼主] 2015-12-29 11:39 clearbug

@ 念念伊梦

o(╯╰╯)o 唉，接触前端还没多长时间，我更是菜鸟啦！最近一段时间也没怎么写，有点懒了。从今天就继续吧，欢迎以后有空来对我写的指点评论哈！

支持(0) 反对(0)

#22楼 2016-01-02 18:38 小丸

博主这篇文章是我看下来讲解js对象最完整全面的文章，感谢博主！！

支持(0) 反对(0)

#23楼[楼主] 2016-01-03 22:06 clearbug

@ 小丸

额额，对你有帮助就好哈！欢迎以后常来指点评论，相互学习。

支持(0) 反对(0)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。

【推荐】超50万VC++源码：大型组态工控、电力仿真CAD与GIS源码库！

【活动】华为云12.12会员节全场1折起 满额送Mate20

【活动】华为云会员节云服务特惠1折起

【活动】腾讯云+社区开发者大会12月15日首都北京盛大起航！