



# 第12课：对象格式深入讲解

■ 主讲老师：万章



## 目录

什么是对象

创建/访问对象的方式

对象的属性的常用操作

对象属性的特性

对象的方法



# 什么是对象

## 什么是对象

对象是属性的无序集合,  
每个属性都是一个名/值  
对.  
属性名称是一个字符串.

属性名称 : 属性值

名/值对

属性名称 : 属性值 , 属性名称 : 属性值

名/值对

名/值对

## 对象的种类

1. 内置对象 (native object) 是由ECMAScript规范定义的对象或类。例如, 数组、函数、日期和正则表达式都是内置对象
2. 宿主对象 (host object) 是由JavaScript解释器所嵌入的宿主环境 (比如Web浏览器) 定义的, 常见的宿主对象有window, document等
3. 自定义对象 (user-defined object) 是由运行中的JavaScript代码创建的对象。



# 创建对象的方式

## 对象直接量

对象直接量是由若干名/值对组成的，名/值对**中间**用冒号分隔，名/值对**之间**用逗号分隔

整个对象用花括号括起来。

属性名可以是JavaScript标识符(即变量符号,如a,b,c)也可以是字符串（包括空字符串, 如:"a","b","c":）。

属性的值可以是任意类型的JavaScript表达式，表达式的值（可以是基本数据类型的值也可以是对象类型的值，就是这个类型的值）

```
var empty = {}; //没有任何属性的对象
```

```
var point = {  
  x: 0,  
  y: 0  
}; //两个属性
```

```
var point2 = {  
  x: point.x,  
  y: point.y + 1  
}; //更复杂的值,x的值是0,y的值是1
```

```
var book = {  
  "main title": "JavaScript", //属性名字里有空格,必须用字符串表示  
  'sub - title': "对象课程", //属性名字里有连字符,必须用字符串表示  
  "for": "基础学员", // "for"是保留字,因此必须用引号  
  author: { //这个属性的值是一个对象  
    firstname: "万", //注意,这里的属性名都没有引号  
    surname: "章"  
  }  
};
```

## 通过new指令创建对象

关键字new后跟随一个函数调用。这里的函数称做构造函数（constructor），构造函数用以初始化一个新创建的对象。

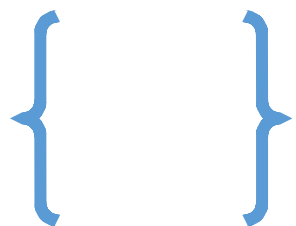
JavaScript语言核心中的原始类型都包含内置构造函数

```
var o = new Object(); // 创建一个空对象，和{}一样  
var a = new Array(); // 创建一个空数组，和[]一样  
var d = new Date(); // 创建一个表示当前时间的Date对象  
var r = new RegExp("js"); // 创建一个可以进行模式匹配的RegExp对象
```



## 对象属性的查询

可以通过点(.)或方括号([])运算符来获取属性的值



### • 属性名称

```
var author = book.author; // 得到book的“author”属性  
var name = author.surname // 得到获得author的“surname”属性
```

点号的左侧一定得是一个对象, 右侧必须是一个以属性名称命名的简单标识符(即变量, 不能使用字符串)

所以一旦元素的某个属性的名称是一个包含了空格的字符串, 就不能使用点号来查询

```
> point  
< {x: 0, y: 0}  
> point.x  
< 0  
> point."x"
```

✖ Uncaught SyntaxError: Unexpected string

```
> |
```

```
> book."for"
```

✖ Uncaught SyntaxError: Unexpected string

```
> book.for  
< "基础学员"
```

```
>
```

## 对象属性的查询

可以通过点(.)或方括号([])运算符来获取属性的值

{ } [ “属性名称” ]

```
var s="main title";  
var title=book["main title"];//得到book的“main title”属性  
var title=book[s];//得到book的“main title”属性
```

方括号的左侧一定得是一个对象, 右侧必须是一个计算结果为字符串()或是可以转换为字符串的值的表达式, 这个字符串就是属性的名字

```
> var o={"0":1}  
< undefined  
> o.0  
✖ Uncaught SyntaxError: Unexpected token  
> o[0]  
< 1  
> o["0"]  
< 1  
> |
```

数字0可转换为字符串0

```
> var o={"addr0":"湖南","addr1":"长沙",addr3:"高新科技园"};  
< undefined  
> o[addr3]  
✖ ▶ Uncaught ReferenceError: addr3 is not defined  
   at <anonymous>:1:3  
> o["addr3"]  
< "高新科技园"  
> o["addr"+3]  
< "高新科技园"  
>
```

字符串" addr" + 3, 的结果是字符串" addr3"

## 对象属性的设置

可以通过点(.)或方括号([])运算符来获取属性的名称结合赋值操作符(=)实现属性值的设置

{ } [ “属性名称” ] = 属性值

{ } • 属性名称 = 属性值

## 对象属性的查询错误

查询一个不存在的属性并不会报错，如果在对象o自身的属性或继承的属性中均未找到属性x，属性访问表达式o.x返回undefined

查询一个不存在的对象的属性，则会直接报错，并提示开发者你所查询的那个对象没有被定义

```
var book = {  
  "main title": "JavaScript", //属性名字里有空格, 必须用字符串表示  
  'sub - title': "对象课程", //属性名字里有连字符, 必须用字符串表示  
  "for": "基础学员", // "for" 是保留字, 因此必须用引号  
  author: { //这个属性的值是一个对象  
    firstname: "万", //注意, 这里的属性名都没有引号  
    surname: "章"  
  }  
};  
book.subtitle; // => undefined: 属性不存在  
book123.subtitle; // book123 is not defined, 直接报错
```



# 对象的属性的常用操作

## 删除对象属性

delete运算符可以删除对象的属性。它的操作数应当是一个属性访问表达式。

```
> var point = {  
    x: 0,  
    y: 0  
}; //两个属性.  
↵ undefined  
> point  
↵ {x: 0, y: 0}  
> delete point.x;  
↵ true  
> point  
↵ {y: 0}  
>
```

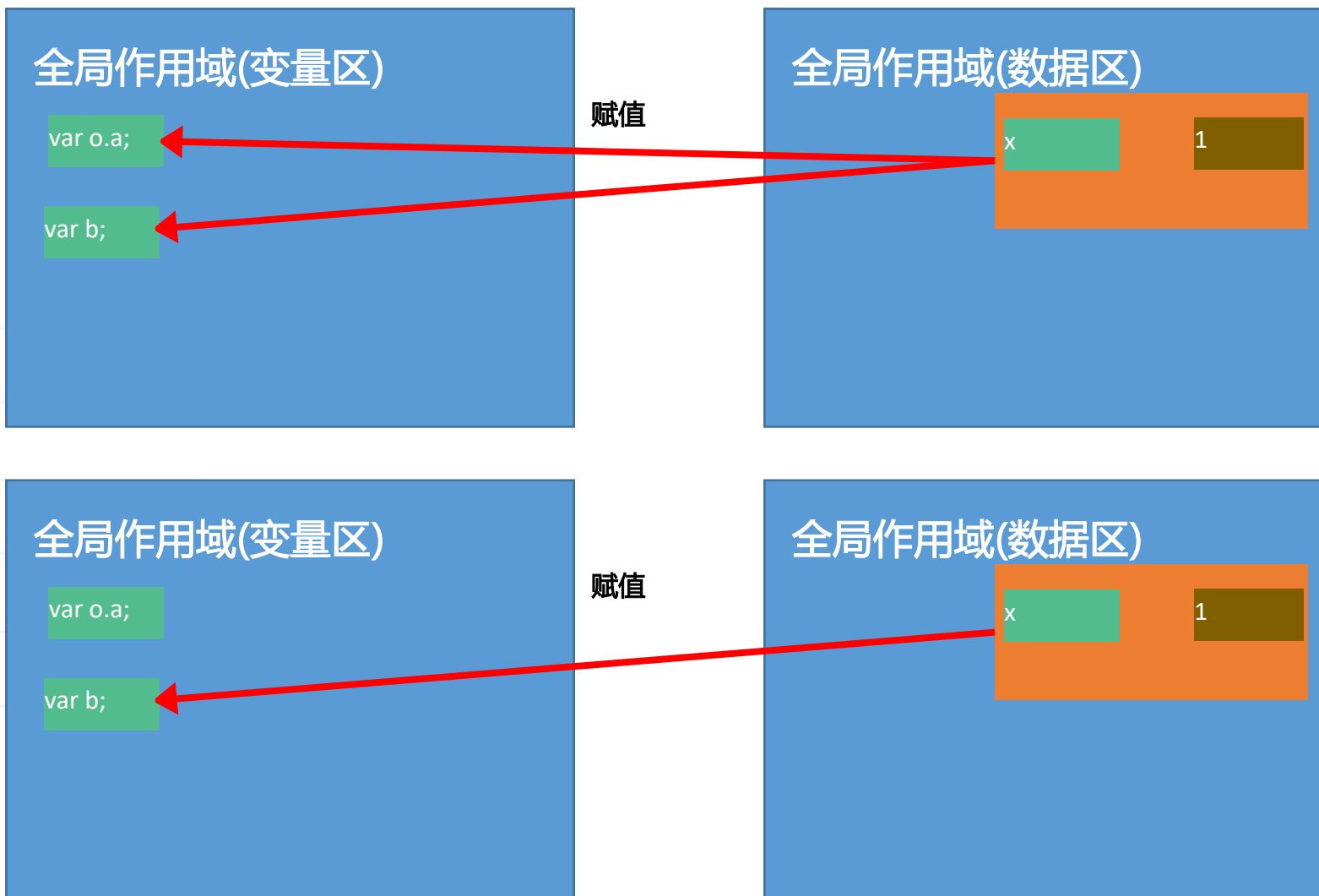
delete { "属性名称" }

delete { • 属性名称 }

# 删除对象属性

需要注意，delete只是断开属性和宿主对象的联系，而不会去操作内存中对象的属性的值的存在与否

```
> var o={a:{x:1}};  
< undefined  
  
> var b=o.a;  
< undefined  
  
> b  
< {x: 1}  
  
> delete o.a;  
< true  
  
> b  
< {x: 1}  
  
> o  
< {}  
  
> |
```



断开了o.a与数据区的联系  
但是b的联系还在

## 检测对象属性之in运算符

我们经常会检测集合中成员的所属关系——判断某个属性是否存在于某个对象中。可以通过in运算符、hasOwnProperty() 方法来完成这个工作，甚至仅通过属性查询也可以做到这一点。

in运算符的左侧是属性名（字符串或是一个能够返回字符串的表达式），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, 不包含则返回false

```
> var point = {  
    x: 0,  
    y: 0  
}; //两个属性.
```

```
< undefined
```

```
> x in point
```

```
✖ ▶ Uncaught ReferenceError: x is not defined  
   at <anonymous>:1:1
```

```
> "x" in point
```

```
< true
```

```
> "z" in point
```

```
< false
```

```
> var c="x";
```

```
< undefined
```

```
> c in point
```

```
< true
```

```
>
```

“属性名称”

in

{ }



## 检测对象属性之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。对于不存在的属性和继承属性它将返回false:

{ } • hasOwnProperty( “属性名称” )

```
var o = {  
  x: 1  
}  
  
o.hasOwnProperty("x"); //true: o有一个自有属性x  
o.hasOwnProperty("y"); //false: o中不存在属性y  
o.hasOwnProperty("toString"); //false: toString是继承属性
```

## 枚举对象属性之propertyIsEnumerable()

除了检测对象的属性是否存在，我们还会经常遍历对象的属性，通常使用for/in循环遍历。for/in循环可以在循环体中遍历对象中所有可枚举的属性（包括自有属性和继承的属性）。

对象继承的内置方法不可枚举的，但在代码中给对象添加的属性都是可枚举的。

## • propertyIsEnumerable (“属性名称”)

```
<script>

var o = {
  x: 1,
  y: 2,
  z: 3
}; //三个可枚举的自有属性

o.propertyIsEnumerable("toString") //=>false, 不可枚举

for (p in o) {
  console.log(p); //输出x、y和z，不会输出toString
} //遍历属性

</script>
```



# 对象的属性的特性

## 对象属性的特性

属性的特性所服务的对象是JavaScript的解析引擎, 是JavaScript引擎的内部值, 所以作为程序员是无法直接获取到的. 在JavaScript中**对象的属性有两种**, 分别是**数据属性**和**访问器属性**.

为了表示特性是内部值, 该规范把它们放在了**两对儿方括号**中, 例如[[Enumerable]]

```
var empty = {}; //没有任何属性的对象
```

```
var point = {  
  x: 0,  
  y: 0  
}; //两个属性
```

```
var point2 = {  
  x: point.x,  
  y: point.y + 1  
}; //更复杂的值,x的值是0,y的值是1
```

**数据属性**包含一个数据值的位置。在这个位置可以读取和写入值。数据属性有4个描述其行为的特性。

**[[Configurable]]**: 表示能否通过**delete** 删除属性从而重新定义属性, 能否修改属性的特性, 或者能否把属性修改为访问器属性。像前面例子中那样直接在对象上定义的属性, 它们的这个特性默认值为**true**。

**[[Enumerable]]**: 表示能否通过**for-in** 循环返回属性。像前面例子中那样直接在对象上定义的属性, 它们的这个特性默认值为**true**。

**[[Writable]]**: 表示能否修改属性的值。像前面例子中那样直接在对象上定义的属性, 它们的这个特性默认值为**true**。

**[[Value]]**: 包含这个属性的数据值。读取属性值的时候, 从这个位置读; 写入属性值的时候, 把新值保存在这个位置。这个特性的默认值为**undefined**。

比如point对象的x属性, 该属性的特性如下

[[Configurable]]:true

[[Enumerable]]:true

[[Writable]]:true

[[Value]]:0

## 获取对象属性的特性Object.getOwnPropertyDescriptor()

Object.getOwnPropertyDescriptor(  , “属性名称”)

```
<script>

var o = {
  x: 1,
  y: 2,
  z: 3
}; //三个可枚举的自有属性

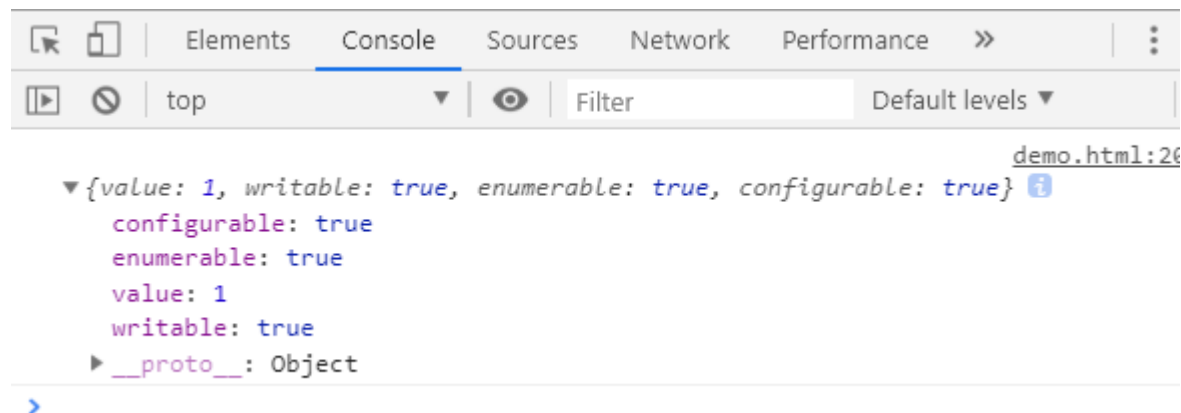
console.log(Object.getOwnPropertyDescriptor(o,"x"));

</script>
```

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，每一个属性都有一个属性描述符，获取方式如上。

从函数名字就可以看

出，Object.getOwnPropertyDescriptor()只能得到自有属性的描述符。要想获得继承属性的特性，需要遍历原型链



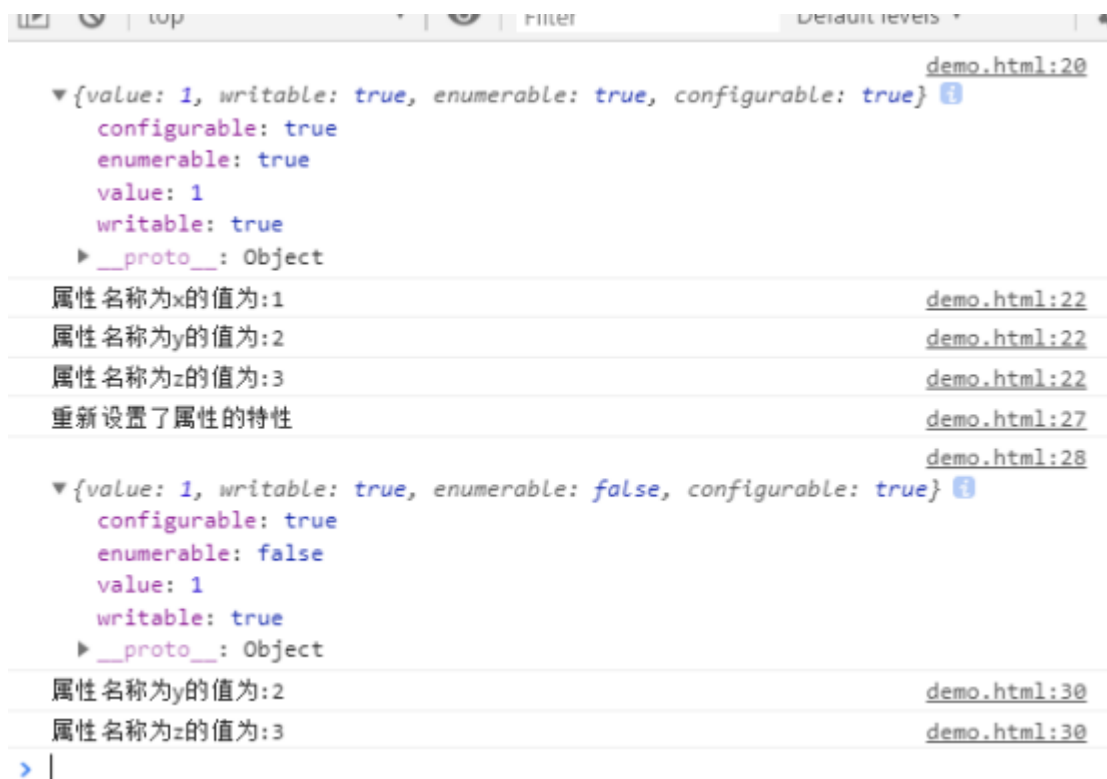
## 设置对象属性的特性Object.defineProperty()

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：

Object.defineProperty( {待修改对象} , “属性名称”, {描述符对象} )

```
var o = {
  x: 1,
  y: 2,
  z: 3
}; //三个可枚举的自有属性

console.log(Object.getOwnPropertyDescriptor(o, "x"));
for (let i in o) {
  console.log("属性名称为" + i + "的值为:" + o[i]); //x,y,z都输出出来了
}
Object.defineProperty(o, "x", {
  enumerable: false
});
console.log("重新设置了属性的特性");
console.log(Object.getOwnPropertyDescriptor(o, "x"));
for (let i in o) {
  console.log("属性名称为" + i + "的值为:" + o[i]); //x不会输出来, 因为x已经无法遍历了
}
```



```
demo.html:20
▼ {value: 1, writable: true, enumerable: true, configurable: true} ⓘ
  configurable: true
  enumerable: true
  value: 1
  writable: true
  ▶ __proto__: Object
属性名称为x的值为:1 demo.html:22
属性名称为y的值为:2 demo.html:22
属性名称为z的值为:3 demo.html:22
重新设置了属性的特性 demo.html:27
demo.html:28
▼ {value: 1, writable: true, enumerable: false, configurable: true} ⓘ
  configurable: true
  enumerable: false
  value: 1
  writable: true
  ▶ __proto__: Object
属性名称为y的值为:2 demo.html:30
属性名称为z的值为:3 demo.html:30
> |
```

## 设置对象属性的特性Object.defineProperty()

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：

Object.defineProperty( {待修改对象} , “属性名称”, {描述符对象} )

```
var o = {
  x: 1,
  y: 2,
  z: 3
}; //三个可枚举的自有属性
console.log(Object.getOwnPropertyDescriptor(o,"x"));
o.x=10;
console.log(o.x);//o.x变成了10
Object.defineProperty(o,"x",{
  writable: false
});
o.x=20;
console.log(o.x);//o.x还是10
console.log(Object.getOwnPropertyDescriptor(o,"x"));
```

```
demo.html:19
▼{value: 1, writable: true, enumerable: true, configurable: true} ⓘ
  configurable: true
  enumerable: true
  value: 1
  writable: true
  ▶ __proto__: Object
10 demo.html:21
10 demo.html:26
demo.html:27
▼{value: 10, writable: false, enumerable: true, configurable: true} ⓘ
  configurable: true
  enumerable: true
  value: 10
  writable: false
  ▶ __proto__: Object
>
```



# 对象的方法

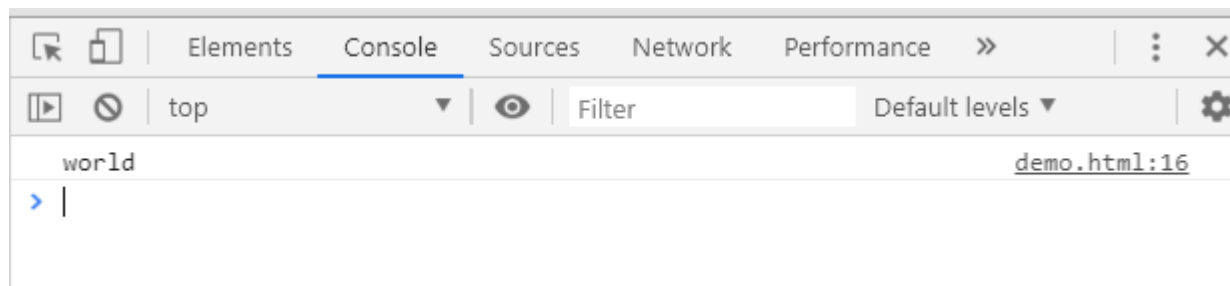


## 对象的方法

对象的属性值的类型可以任意, 如果某个对象的属性的值的类型是函数的话, 那么我们把这个值为函数的属性名称称之为对象的方法  
典型代表有.toString();

```
var o = {  
  x: "hello",  
  y: function (str) {  
    console.log(str);  
  }  
}  
o.y("world");
```

属性y是对象o的方法



## 对象的方法之this

函数中的this 对象是在运行时基于函数的执行环境绑定的：在全局函数中，this 等于window，而当函数被作为某个对象的方法调用时，this 等于那个对象。不过，匿名函数的执行环境具有全局性，因此其this 对象通常指向window。

```
var o = {  
  x: "hello",  
  y: function (str) {  
    console.log(this); // 输出对象o  
  }  
}  
o.y();  
  
function fn() {  
  console.log(this); // 输出对象window  
}  
fn();
```

▶ {x: "hello", y: f}

[demo.html:16](#)

▶ Window {postMessage: f, blur: f, focus: f, close: f, parent: Window, ...}

[demo.html:22](#)

>



## 对象的方法之this

其他代码中的this指的是, 该行代码的运行环境

## 对象访问器属性之getter和setter

访问器属性不包含数据值；它们包含一对儿getter 和setter 函数（不过，这两个函数都不是必需的）。在读取访问器属性时，会调用getter 函数，这个函数负责返回有效的值；在写入访问器属性时，会调用setter 函数并传入新值，这个函数负责决定如何处理数据

**访问器属性**有如下4 个特性。

**[[Configurable]]**：表示能否通过delete 删除属性从而重新定义属性，能否修改属性的特性，或者能否把属性修改为数据属性。对于直接在对象上定义的属性，这个特性的默认值为true。

**[[Enumerable]]**：表示能否通过for-in 循环返回属性。对于直接在对象上定义的属性，这个特性的默认值为true。

**[[Get]]**：在读取属性时调用的函数。默认值为undefined。

**[[Set]]**：在写入属性时调用的函数。默认值为undefined。

访问器属性不能直接定义，必须使用Object.defineProperty()来定义

## 对象访问器属性之getter和setter

```
var book = {  
  _year: 2004, // _year 前面的下划线是一种常用的记号, 用于表示只能通过对象方法访问的属性。  
  edition: 1  
};  
  
Object.defineProperty(book, "year", {  
  get: function () {  
    return this._year;  
  },  
  set: function (newValue) {  
    if (newValue > 2004) {  
      this._year = newValue;  
      this.edition += newValue - 2004;  
    }  
  },  
  enumerable: true,  
  configurable: true  
});  
  
book.year = 2005;  
console.log(book.edition); //2
```