

《手机》 移动技术高级开发

3 ES5.1 之 JavaScript 基础



CONTENS



- 1 JavaScript 简介
- 2 JavaScript 数据类型
- 3 JavaScript 标准库
- 4 JavaScript 面向对象编程
- 5 参考资料



1

—PART 01—

JavaScript 简介

▶ 导论 ▶ 历史 ▶ 基本语法



什么是 JavaScript 语言？



JavaScript 是一种轻量级的脚本语言。

所谓“脚本语言”（script language），指的是它不具备开发操作系统的能力，而是只用来编写控制其他大型应用程序（比如浏览器）的“脚本”。

JavaScript 也是一种嵌入式（embedded）语言。

它本身提供的核心语法不算很多，只能用来做一些数学和逻辑运算。JavaScript 本身不提供任何与 I/O（输入/输出）相关的 API，都要靠宿主环境（host）提供，所以 JavaScript 只合适嵌入更大型的应用程序环境，去调用宿主环境提供的底层 API。

目前，已经嵌入 JavaScript 的宿主环境有多种，最常见的环境就是浏览器，另外还有服务器环境，也就是 Node 项目。

从语法角度看，JavaScript 语言是一种“对象模型”语言。

各种宿主环境通过这个模型，描述自己的功能和操作接口，从而通过 JavaScript 控制这些功能。但是，JavaScript 并不是纯粹的“面向对象语言”，还支持其他编程范式（比如函数式编程）。这导致几乎任何问题，JavaScript 都有多种解决方法。

JavaScript 语言的显著特点 (1/2)

操控浏览器的能力

JavaScript 的发明目的，就是作为浏览器的内置脚本语言，为网页开发者提供操控浏览器的能力。它是目前唯一一种通用的浏览器脚本语言，所有浏览器都支持。它可以让网页呈现各种特殊效果，为用户提供良好的互动体验。

广泛的使用领域

(1) 浏览器的平台化；(2) Node；(3) 数据库操作；(4) 移动平台开发；(5) 内嵌脚本语言；(6) 跨平台的桌面应用程序。

可以预期，JavaScript 最终将能让你只用一种语言，就开发出适应不同平台（包括桌面端、服务器端、手机端）的程序。早在2013年9月的[统计](#)之中，JavaScript 就是当年 GitHub 上使用量排名第一的语言。

易学性

Any application that can be written in JavaScript will eventually be written in JavaScript.

所有可以用 JavaScript 编写的程序，最终都会出现 JavaScript 的版本。

——Jeff Atwood (stackoverflow创始人)

JavaScript 语言的显著特点 (2/2)

强大的性能

(1) **灵活的语法，表达力强。**JavaScript 既支持类似 C 语言清晰的过程式编程，也支持灵活的函数式编程，可以用来写并发处理（concurrent）。这些语法特性已经被证明非常强大，可以用于许多场合，尤其适用异步编程。

(2) **支持编译运行。**JavaScript 语言本身，虽然是一种解释型语言，但是在现代浏览器中，JavaScript 都是编译后运行。程序会被高度优化，运行效率接近二进制程序。而且，JavaScript 引擎正在快速发展，性能将越来越好。

(3) **事件驱动和非阻塞式设计。**JavaScript 程序可以采用事件驱动（event-driven）和非阻塞式（non-blocking）设计，在服务器端适合高并发环境，普通的硬件就可以承受很大的访问量。

开放性

JavaScript 是一种开放的语言。它的标准 ECMA-262 是 ISO 国际标准，写得非常详尽明确；该标准的主要实现都是开放的，而且质量很高。这保证了这门语言不属于任何公司或个人，不存在版权和专利的问题。

社区支持和就业机会

全世界程序员都在使用 JavaScript，它有着极大的社区、广泛的文献和图书、丰富的代码资源。绝大部分你需要用到的功能，都有多个开源函数库可供选用。

作为项目负责人，你不难招聘到数量众多的 JavaScript 程序员；作为开发者，你也不难找到一份 JavaScript 的工作。

JavaScript 语言的历史

JavaScript 与 Java 的关系

JavaScript 的基本语法和对象体系，是模仿 Java 而设计的。但是，JavaScript 没有采用 Java 的静态类型。正是因为 JavaScript 与 Java 有很大的相似性，所以这门语言才从一开始的 LiveScript 改名为 JavaScript。基本上，JavaScript 这个名字的原意是“很像Java的脚本语言”。

JavaScript 语言的函数是一种独立的数据类型，以及采用基于原型对象（prototype）的继承链。这是它与 Java 语法最大的两点区别。JavaScript 语法要比 Java 自由得多。

另外，Java 语言需要编译，而 JavaScript 语言则是运行时由解释器直接执行。

总之，JavaScript 的原始设计目标是一种小型的、简单的动态语言，与 Java 有足够的相似性，使得使用者（尤其是 Java 程序员）可以快速上手。

JavaScript 与 ECMAScript 的关系

1996年11月，Netscape 公司决定将 JavaScript 提交给国际标准化组织 ECMA（European Computer Manufacturers Association），希望 JavaScript 能够成为国际标准。

1997年7月，ECMA 组织发布262号标准文件（ECMA-262）的第一版，规定了浏览器脚本语言的标准，并将这种语言称为 ECMAScript。

ECMAScript 和 JavaScript 的关系是，前者是后者的规格，后者是前者的一种实现。在日常场合，这两个词是可以互换的。

JavaScript 的版本

1997年7月，ECMAScript 1.0发布。

1998年6月，ECMAScript 2.0版发布。

1999年12月，ECMAScript 3.0版发布。成为 JavaScript 的通行标准，得到了广泛支持。

.....

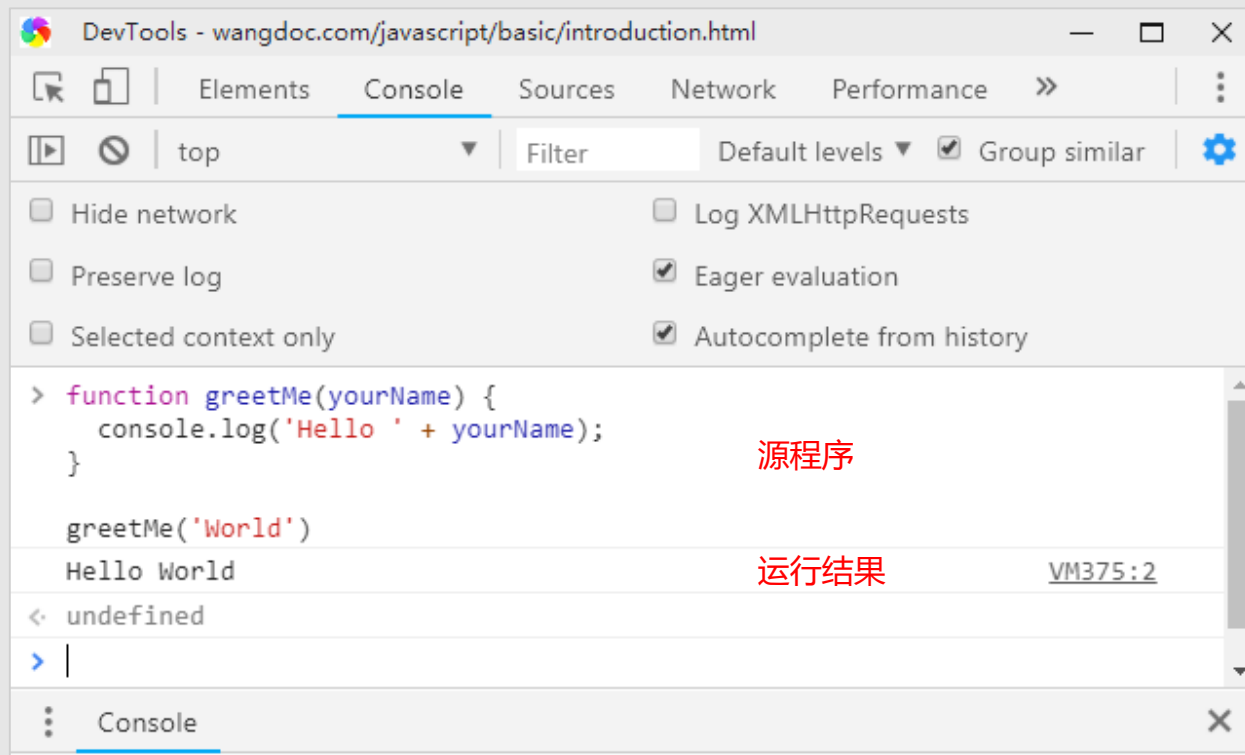
2011年6月，ECMAScript 5.1版发布，并且成为 ISO 国际标准 (ISO/IEC 16262:2011) 。到了2012年底，所有主要浏览器都支持 ECMAScript 5.1版的全部功能。

2015年6月，ECMAScript 6 正式发布，并且更名为 “ECMAScript 2015” 。

实验环境

推荐：Chrome 浏览器 → 开发者工具 → 控制台

快捷键：Ctrl + Shift + J



基本语法

标识符命名规则

第一个字符，可以是任意 Unicode 字母（包括英文字母和其他语言的字母），以及美元符号（\$）和下划线（_）。

第二个字符及后面的字符，除了 Unicode 字母、美元符号和下划线，还可以用数字0-9。

保留字不能用作标识符。

```
1.  arg0    // 合法
2.  _tmp    // 合法
3.  $elem   // 合法
4.  π       // 合法
5.
6.  1a      // 第一个字符不能是数字
7.  23      // 同上
8.  ***     // 标识符不能包含星号
9.  a+b     // 标识符不能包含加号
10. -d      // 标识符不能包含减号或连词线
11.
12. var 临时变量 = 1; // 合法
```

JavaScript的很多语法与C语言相同，学习时应注意和其他语言语法有差异的地方。

2

PART 02

JavaScript 数据类型

- ▶ 概述
- ▶ null, undefined 和布尔值
- ▶ 数值
- ▶ 字符串
- ▶ 对象
- ▶ 函数
- ▶ 数组



概述

数值 (number)

整数和小数（比如1和3.14），三大原始类型之一。

字符串 (string)

文本（比如Hello World），三大原始类型之一。

布尔值 (boolean)

表示真伪的两个特殊值，即true（真）和false（假），三大原始类型之一。

Undefined

表示“未定义”或不存在，即由于目前没有定义，所以此处暂时没有任何值

Null

表示空值，即此处的值为空。

对象 (object)

各种值组成的集合。

Symbol

表示独一无二的值。ES6引入。

null, undefined

null表示空值

即该处的值现在为空。调用函数时，某个参数未设置任何值，这时就可以传入null，表示该参数为空。比如，某个函数接受引擎抛出的错误作为参数，如果运行过程中未出错，那么这个参数就会传入null，表示未发生错误。

undefined表示“未定义”

布尔值

转换规则是除了下面六个值被转为false，其他值都视为true。

- undefined
- null
- false
- 0
- NaN
- ""或''（空字符串）

```
1.  if (') {  
2.      console.log('true');  
3.  }  
4.  // 没有任何输出  
5.  
6.  if ([]) {  
7.      console.log('true');  
8.  }  
9.  // true  
10.  
11. if ({}) {  
12.     console.log('true');  
13. }  
14. // true  
15.  
16.  
17.
```

数值

所有数字都是以64位浮点数形式储存，即使整数也是如此。 第1行

由于浮点数不是精确的值，所以涉及小数的比较和运算要特别小心。 第3-5行

特殊数值：

NaN表示“非数字” 第7行

Infinity表示“无穷”

与数值相关的全局方法 第9-15行

`parseInt()` 方法用于将字符串转为整数，进制转换。

`parseFloat()` 方法用于将一个字符串转为浮点数。

`isNaN()` 方法可以用来判断一个值是否为NaN。

`isFinite()` 表示某个值是否为正常的数值。

```
1. 1 === 1.0 // true
2.
3. 0.1 + 0.2 === 0.3 // false
4. 0.3 / 0.1 // 2.9999999999999996
5. (0.3 - 0.2) === (0.2 - 0.1) // false
6.
7. NaN === NaN // false
8.
9. parseInt('123') // 123
10. parseInt('1000', 2) // 8
11. parseFloat('3.14') // 3.14
12. isNaN(NaN) // true
13. isNaN(123) // false
14. isFinite(Infinity) // false
15. isFinite(-1) // true
```


字符串

定义

字符串就是零个或多个排在一起的字符，放在单引号或双引号之中。 (第1-2行)

单引号字符串的内部，可以使用双引号。 (第3行)

双引号字符串的内部，可以使用单引号。 (第4行)

注意事项

建议JavaScript 语言的字符串只使用单引号。

字符串内部的单个字符无法改变和增删。 (第14-16行)

length属性返回字符串的长度。 (第17行)

```
1. 'abc' // 合法
2. "abc" // 合法
3. 'key = "value"' // 合法
4. "It's a long journey" // 合法
5.
6. var s = 'hello';
7. s[0] // "h"
8. s[1] // "e"
9. s[4] // "o"
10. 'abc'[0] // "a"
11. 'abc'[3] // undefined
12. 'abc'[-1] // undefined
13.
14. var s = 'hello';
15. s[1] = 'a';
16. s // "hello"
17. s.length // 5
```

对象 (1/2) : 一组“键值对” (key-value) 的集合。

属性

键名都是字符串，加不加引号都可以。 (第1-4行)

如果键名不符合标识名的条件，则必须加上引号。

如果属性的值还是一个对象，就形成了链式引用。 (第6-9行)

属性可以动态创建，不必在对象声明时就指定。 (第8行)

对象还是代码块

无法确定是对象还是代码块，一律解释为代码块。 (第11行)

如果要解释为对象，最好在大括号前加上圆括号。

eval语句：作用是对字符串求值。 (第15-16行)

```
1. var obj = {  
2.     foo: 'Hello',  
3.     'bar': 'World'  
4. };  
5.  
6. var o1 = {};  
7. var o2 = { bar: 'hello' };  
8. o1.foo = o2;  
9. o1.foo.bar // "hello"  
10.  
11. { console.log(123) } // 123, 代码块  
12. ({ foo: 123 }) // 正确, 对象  
13. ({ console.log(123) }) // 报错, 对象  
14.  
15. eval('{foo: 123}') // 123  
16. eval('({foo: 123})') // {foo: 123}
```

对象 (2/2)

属性的操作

属性的读取：使用点运算符和方括号运算符 (第1-5行)

属性的赋值：使用点运算符和方括号运算符 (第7-9行)

属性的查看：Object.keys() (第10行)

属性的删除：delete 命令 (第11-12行)

属性是否存在：in 运算符, hasOwnProperty() (第13-14行)

属性的遍历：for...in 循环 (第16-20行)

操作同一个对象的多个属性：with语句 (第22-27行)

```
1.  var obj = {
2.      p: 'Hello World'
3.  };
4.  obj.p // "Hello World"
5.  obj['p'] // "Hello World"
6.
7.  var obj = {};
8.  obj.foo = 'Hello';
9.  obj['bar'] = 'World';
10. Object.keys(obj); // ['foo', 'bar']
11. delete obj.foo // true
12. Object.keys(obj) // ['bar']
13. 'bar' in obj //true
14. obj.hasOwnProperty('toString') //false
15.
16. var obj = {a: 1, b: 2, c: 3};
17. for (var i in obj) {
18.     console.log('键名: ', i);
19.     console.log('键值: ', obj[i]);
20. }
21.
22. // 例一
23. var obj = { p1: 1, p2: 2, };
24. with (obj) { p1 = 4; p2 = 5; }
25. // 等同于
26. obj.p1 = 4;
27. obj.p2 = 5;
```

函数

函数的声明

- (1) function 命令 第1-4行
- (2) 函数表达式 第6-9行
- (3) Function 构造函数 (不推荐) 第11-20行

```
1. // 语法一: 使用 function 命令声明函数
2. function print(s) {
3.     console.log(s);
4. }
5.
6. // 语法二: 使用函数表达式声明函数
7. var print = function(s) {
8.     console.log(s);
9. };
10.
11. // 语法三: 使用Function构造函数声明函数 (不推荐)
12. var add = new Function(
13.     'x',
14.     'y',
15.     'return x + y' // 最后一个参数是函数体
16. );
17. // 等同于
18. function add(x, y) {
19.     return x + y;
20. }
```

函数

函数的属性和方法

(1) name属性返回函数的名字。第1-6行

(2) length属性返回函数定义的参数个数。第8-9行

(3) toString方法返回函数的源码。第11-16行

```
1.  function f1() {}
2.  f1.name // "f1"
3.  var f2 = function () {};
4.  f2.name // "f2"
5.  var f3 = function myName() {};
6.  f3.name // 'myName'
7.
8.  function f(a, b) {}
9.  f.length // 2
10.
11. function f() {
12.     a();
13.     b();
14. }
15. f.toString()
16. // 'function f() {\n  a();\n  b();\n}'
```

函数

函数的参数

(1) 常规用法。

(2) 函数参数不是必需的。

(3) 传递方式：

参数是原始类型：传值传递

参数是复合类型：传址传递

(4) arguments 对象

```
1. function square(x) {  
2.     return x * x;  
3. }  
4. square(2) // 4  
5. square(3) // 9
```

(1)

```
1. function f(a, b) {  
2.     return a;  
3. }  
4. f(1, 2, 3) // 1  
5. f(1) // 1  
6. f() // undefined  
7. f.length // 2
```

(2)

```
1. var p = 2;  
2. function f(p) {  
3.     p = 3;  
4. }  
5. f(p);  
6. p // 2  
7.  
8. var obj = { p: 1 };  
9. function f(o) {  
10.    o.p = 2;  
11. }  
12. f(obj);  
13. obj.p // 2
```

(3)

```
1. var f = function (one) {  
2.     console.log(arguments[0]);  
3.     console.log(arguments[1]);  
4.     console.log(arguments[2]);  
5. }  
6.  
7. f(1, 2, 3)  
8. // 1  
9. // 2  
10. // 3  
11.  
12.  
13.  
14.
```

(4)

函数

闭包

(1) 变量作用域：全局作用域和函数作用域。

(2) 闭包：能够读取其他函数内部变量的函数。

(3) 闭包的用处：①可以使得函数内部变量一直存在；②封装对象的私有属性和私有方法。

```
1. function f1() {  
2.   var n = 999;  
3.   function f2() {  
4.     console.log(n);  
5.   }  
6.   return f2;  
7. }  
8. var result = f1();  
9. result(); // 999  
10. // 其中f2就是闭包 (2)
```

```
1. var n = 999;  
2.  
3. function f1() {  
4.   console.log(n);  
5. }  
6. f1() // 999  
(1.全局作用域)
```

```
1. function f1() {  
2.   var n = 999;  
3. }  
4.  
5. console.log(n)  
6. // Uncaught ReferenceError:  
7. n is not defined (1.函数作用域)
```

```
1. function createIncrementor(start) {  
2.   return function () {  
3.     return start++;  
4.   };  
5. }  
6.  
7.  
8. var inc = createIncrementor(5);  
9. entor(5);  
10.  
11. inc() // 5  
12. inc() // 6  
13. inc() // 7  
(3)
```

```
1. function Person(name) {  
2.   var _age;  
3.   function setAge(n) {  
4.     _age = n;  
5.   }  
6.   function getAge() {  
7.     return _age;  
8.   }  
9.   return {  
10.    name: name,  
11.    getAge: getAge,  
12.    setAge: setAge  
13.  };  
14. }  
15.  
16. var p1 = Person('张三');  
17. p1.setAge(25);  
18. p1.getAge() // 25  
(3)
```

数组

(1) 定义。 (第1-14行)

(2) length属性。 (第16行)

(3) in运算符。 (第18-20行)

(4) for...in (第22-25行)

(5) forEach (第27-30行)

```
1.  var arr = ['a', 'b', 'c'];
2.
3.  var arr = [];
4.  arr[0] = 'a';
5.  arr[1] = 'b';
6.
7.  var arr = [
8.      {a: 1},
9.      [1, 2, 3],
10.     function() {return true;}
11. ];
12. arr[0] // Object {a: 1}
13. arr[1] // [1, 2, 3]
14. arr[2] // function () {return true;}
15.
16. ['a', 'b', 'c'].length // 3
17.
18. var arr = [ 'a', 'b', 'c' ];
19. 2 in arr // true
20. 4 in arr // false
21.
22. var a = [1, 2, 3];
23. for (var i in a) {
24.     console.log(a[i]);
25. }
26.
27. var colors = ['red', 'green', 'blue'];
28. colors.forEach(function (color) {
29.     console.log(color);
30. });
```


3

PART 03

JavaScript 标准库

- ▶ Object 对象
- ▶ 属性描述对象
- ▶ Array 对象
- ▶ 包装对象
- ▶ Boolean 对象
- ▶ Number 对象
- ▶ String 对象
- ▶ Math 对象
- ▶ Date 对象
- ▶ RegExp 对象
- ▶ JSON 对象



本节内容省略，请大家自行学习课件最后一页的参考资料。

4

—PART 04—

JavaScript 面向对象编程

▶ 实例对象

▶ 对象的继承



Object 对象

概述

第1-9行

JavaScript 的所有其他对象都继承自Object对象。

Object()函数

第11-13行

将任意值转为对象。

Object构造函数

第15-17行

直接通过它来生成新对象。

Object的静态方法

第19-20行

指部署在Object对象自身的方法。

Object.keys(), Object.getOwnPropertyNames(), .

Object的实例方法

第22-23行

定义在Object.prototype对象的方法。

Object.prototype.valueOf(), Object.prototype.toString(),

```
1.  Object.print = function (o) {
2.      console.log(o)
3.  };
4.
5.  Object.prototype.print = function () {
6.      console.log(this);
7.  };
8.  var obj = new Object();
9.  obj.print() // Object
10.
11. var obj = Object(1);
12. obj instanceof Object // true
13. obj instanceof Number // true
14.
15. var o1 = {a: 1};
16. var o2 = new Object(o1);
17. o1 === o2 // true
18.
19. var obj = { p1: 123, p2: 456 };
20. Object.keys(obj) // ["p1", "p2"]
21.
22. var o1 = new Object();
23. o1.toString() // "[object Object]"
```

实例对象

构造函数 第1-5行

JavaScript语言的对象体系，不是基于“类”的，而是基于构造函数（constructor）和原型链（prototype）。

构造函数名字的第一个字母通常大写。

生成对象的时候，必须使用new命令。

函数体内部使用this关键字代表所要生成的对象实例。

new 命令的原理 第7-10行

使用new命令时，它后面的函数依次执行下面的步骤。

1. 创建一个空对象，作为将要返回的对象实例。
2. 将这个空对象的原型，指向构造函数的prototype属性。
3. 将这个空对象赋值给函数内部的this关键字。
4. 开始执行构造函数内部的代码。

Object.create() 第12-20行

生成新的实例对象，新对象关联模板对象。

```
1.  var Vehicle = function () {
2.      this.price = 1000;
3.  };
4.  var v = new Vehicle();
5.  v.price // 1000
6.
7.  var Vehicle = function (p) {
8.      this.price = p;
9.  };
10. var v = new Vehicle(500);
11.
12. var person1 = {
13.     name: '张三',
14.     greeting: function() {
15.         console.log('Hi! ' + this.name);
16.     }
17. };
18. var person2 = Object.create(person1);
19. person2.name // 张三
20. person2.greeting() // Hi! 张三.
```

对象的继承

JavaScript 语言的继承不通过 `class`，而是通过“原型对象”（`prototype`）实现。

构造函数的缺点

同一个构造函数的两个实例，相同的行为会生成两次。
第3、9行

解决方法：JavaScript 的原型对象（`prototype`）。

第14、19行

原型对象的作用

定义所有实例对象共享的属性和方法。

原型链

所有对象都有自己的原型对象（`prototype`）；

任何一个对象，都可以充当其他对象的原型；

原型对象也是对象，所以它也有自己的原型；

“原型链”：对象到原型，再到原型的原型……

原型链的尽头就是 `null`。

```
1. function Animal(name) {  
2.   this.name = name;  
3.   this.walk = function () {  
4.     console.log(this.name+' is walking');  
5.   };  
6. }  
7. var cat1 = new Animal('大毛');  
8. var cat2 = new Animal('二毛');  
9. cat1.walk === cat2.walk; // false  
10.  
11. function Animal(name) {  
12.   this.name = name;  
13. }  
14. Animal.prototype.walk = function () {  
15.   console.log(this.name+' is walking');  
16. };  
17. var cat1 = new Animal('大毛');  
18. var cat2 = new Animal('二毛');  
19. cat1.walk === cat2.walk; // true
```

两种继承模型 (1/2)

原型继承模型

```
1. function Foo(who) {
2.     this.me = who;
3. }
4. Foo.prototype.identify = function() {
5.     return 'I am ' + this.me;
6. };
7.
8. function Bar(who) {
9.     Foo.call(this, who);
10. }
11. Bar.prototype = Object.create(Foo.prototype);
12.
13. Bar.prototype.speak = function() {
14.     console.log('Hello, ' + this.identify() + '.');
15. };
16.
17. var b1 = new Bar('b1');
18. var b2 = new Bar('b2');
19.
20. b1.speak();
21. b2.speak();
```

两种继承模型 (2/2)

对象关联模型

```
1.  Foo = {
2.      init: function(who) {
3.          this.me = who;
4.      },
5.      identify: function() {
6.          return 'I am ' + this.me;
7.      }
8.  };
9.  Bar = Object.create(Foo);
10.
11. Bar.speak = function() {
12.     console.log('Hello, ' + this.identify() + '.');
13. };
14.
15. var b1 = Object.create(Bar);
16. b1.init('b1');
17. var b2 = Object.create(Bar);
18. b2.init('b2');
19.
20. b1.speak();
21. b2.speak();
```


5

PART 05

参考资料



运算符

数据类型的转换

错误处理机制

编程风格

异步操作

DOM

事件

浏览器模型

网道 / WangDoc.com

JavaScript 教程

网道 (WangDoc.com) , 互联网文档计划


本教程全面介绍 JavaScript 核心语法, 从最简单的讲起, 循序渐进、由浅入深, 力求清晰易懂。所有章节都带有大量的代码实例, 便于理解和模仿, 可以用到实际项目中, 即学即用。

本教程适合初学者当作 JavaScript 语言入门教程, 也适合当作日常使用的参考手册。

入门篇 

本教程采用[知识共享 署名-相同方式共享 3.0协议](#)。

分享本文     

 JavaScript 教程

- ▶ 入门篇
- ▶ 数据类型
- ▶ 运算符
- ▶ 语法专题
- ▶ 标准库
- ▶ 面向对象编程
- ▶ 异步操作
- ▶ DOM
- ▶ 事件
- ▶ 浏览器模型
- ▶ 附录: 网页元素接口

自学JavaScript教程: <https://wangdoc.com/javascript/index.html>