Expert
JavaScript

Apress®

JavaScript 专家编程

[美] Mark Daggett 著
刘尚奇 张久坤 魏兆玉 译

- 深入学习JavaScript，构建更好的应用

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

Expert
JavaScript

JavaScript 专家编程

[美] Mark Daggett 著
刘尚奇 张久坤 魏兆玉 译

人民邮电出版社
北京

图书在版编目（CIP）数据

JavaScript专家编程 /（美）达格特（Daggett,M.）著；刘尚奇，张久坤，魏兆玉译. -- 北京：人民邮电出版社，2015.8
ISBN 978-7-115-39276-3

Ⅰ. ①J… Ⅱ. ①达… ②刘… ③张… ④魏… Ⅲ. ①JAVA语言－程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2015)第146382号

内 容 提 要

JavaScript 是一种脚本语言，已广泛用于 Web 应用开发。本书引导读者深入学习 JavaScript，并成为 JavaScript 专家。

全书共 10 章，分别介绍了对象和原型、函数、闭包、术语、异步编程、JavaScript 的 IRL、编程风格、工作流、代码质量、提高可测试性等内容。通过对一系列内容和示例的讲解，本书进一步剖析了 JavaScript 的内部机制，为读者呈现更加全面的 JavaScript。

本书适合有一定经验的 JavaScript 开发人员阅读，能够帮助读者更好地认识和运用 JavaScript 语言。

版 权 声 明

◆ 著　［美］Mark Daggett
译　刘尚奇　张久坤　魏兆玉
责任编辑　陈冀康
责任印制　张佳莹　焦志炜
◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路 11 号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京鑫正大印刷有限公司印刷
◆ 开本：800×1000　1/16
印张：13.5
字数：324 千字　2015 年 8 月第 1 版
印数：1－3 500 册　2015 年 8 月北京第 1 次印刷
著作权合同登记号　图字：01-2014-6954 号
定价：49.00 元
读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315

作者简介

■ Mark Daggett 是一名具有开拓精神的新媒体人，专业的开发人员，还在堪萨斯城的一个社会创新实验室 Humansized 任 CEO。同时，他是 Pledgie.com（一个流行的众筹平台）的合伙创始人，也是艺术学的兼职教授。他曾在用户体验和用户交互设计、概念开发以及战略规划领域工作了近 20 年，担任过各种风投、咨询以及顾问的角色。他是洛克菲勒新媒体奖提名人，并曾经被《纽约时报》（New York Times）、《世界报》（Le Monde）、《连线》杂志（WIRED Magazine）以及《表面》杂志（Surface Magazine）等报道过。Mark 的个人网站是 http://www.markdaggett.com，Twitter 账号是 @ heavysixer。

技术审校简介

■ Jonathan Fielding 是英国的一名 Web 开发人员，在营销行业中担任高级开发人员。他是响应式开源项目——SimpleStateManager 的首席开发人员，同时也经常为其他各种各样的开源项目贡献代码。

前　言

在我看来，好的技术书籍是磁带、藏宝图和现场札记内容的混合体。本书就是我灌注了很多心血而将这些不同形式融为一体的一本书。

老一辈的人还记得，磁带内容是由很多歌曲组成的。这些磁带经常被作为礼物送给朋友、恋人。人们会挑选一些个人喜欢的歌，或者围绕某个主题组织在一起的歌，录入这盘磁带中。通常，当听磁带的人听到这些歌时，这些歌就会勾起人们对录制者的记忆。这本书就是一盘我录给你们的 JavaScript 方面的磁带。这些章节包括 JavaScript 中我喜欢的一些方面，也包括不容易被理解的主题，因为这些主题在 tweet 或博客中不容易解释清楚。这本书给了这些主题足够的空间，可以细致的进行阐述。

在我小的时候，我发现根据藏宝图寻宝是一个很有趣的过程。任何人只要沿着某个地图走，就会变得富有，我被这个想法迷住了。这本书虽然不会带你找到埋藏的宝藏，但你也会有所收获。我把这个语言的内部工作原理设计成一张藏宝图，你可以跟着这张藏宝图找到最后的宝藏。通过跟我一起挖掘 JavaScript 中的这些概念，最后你会对 JavaScript 有更深入的了解。

现场札记一般由科学家撰写。他们会把他们在某个现场的想法、观察到的东西以及与他们研究方向相关的直觉都记录下来。他们甚至在札记中夹杂一些现场的树叶、花瓣或者其他一些天然的东西。札记是一个针对某个特定主题中的某个观点过滤出的高度上下文相关的日志。现场札记的目的是当科学家们不在现场时，还仍然能够对当时的现场进行深入研究。

本书是我的关于 JavaScript 的现场札记。我会用它来帮助我记忆和理解语言中的特色。我推荐你也试一下这个方法，你可以在这本书边缘处打些草稿，高亮某些部分，或者夹一些书签。这不是一本需要珍藏的书，而希望成为一本通过你的使用而不断完善的“活笔记”。

目　录

第1章

对象和原型

练习不会造就完美，只有使用最佳的方法来练习才能造就完美。

——Vince Lombardi

对专家来说，把 JavaScript 的核心概念讲上 3 章似乎有点多，毕竟这些是语言最基本的组成部分。我的主张是，有的人虽然不能读写，但可以说话。就像有的开发人员对 JavaScript 的基本功能很熟悉，但对里面那些复杂的东西可能就没那么了解了。

本书的目标是像明灯一样照亮语言中那些晦涩的角落。里面包含的很多概念你可能已经试着学习过了，甚至可以假设你已经理解了。这里可以想象一下：你正降落到你大脑中储存着 JavaScript 的那一个房间。本书可以被看作是一盏探照灯，用来检查你的 JavaScript 根基中那些有裂纹的地方。本章和下面的章节是用来弥补你的 JavaScript 知识漏洞的。不要觉得回顾这些知识没用，这其实是对 JavaScript 知识结构的重新梳理。

我会先高度概括一下 JavaScript 的语言目标。但你知道它之前，放平你的肚子，匍匐穿过 JavaScript 中那些鲜为人知的概念。我会首先详细介绍跟对象和原型相关的重要思想，然后在接下来的章节中介绍函数和闭包，这些都是 JavaScript 的基础。

1.1 鸟瞰 JavaScript

我们所说的 JavaScript 实际上是 ECMAScript 语言规范的一个实现。JavaScript 若想被看作实现 ECMAScript 规范的一个有效版本，它必须支持规范中定义的语法和语义。作为 ECMAScript 的实现，JavaScript 必须给程序员提供可使用的多种类型（types）、属性（properties）、值（values）、函数（funtion）和一些保留字（reserved words）。

一旦 JavaScript 的某个版本已经符合了 ECMAScript 规范，那么语言设计者就可以自由地对版本进行加强，加入他们认为合适的额外功能和方法。ECMAScript 规范中明确说明允许这样的扩展，正如下面读到的：

符合标准的 ECMAScript 实现，允许提供超出本规范描述的额外类型、值、对象、属性和函数。尤其是本规范中描述的对象，允许提供未在本规范中描述的属性和值。一个合乎 ECMAScript 规范的实现允许加入没有在规范中描述的程序语法和正则表达式语法。

在 ECMAScript 中，一些额外的特性可以与核心要素并行存在，但仍然被认为是一种有效的 ECMAScript 标准的实现，这是 ECMAScript 标准组织发展的一个标志。ECMAScript 对特性的要求比较宽泛，这带来了一些好处但也有弊端。虽然灵活地添加新功能，可以鼓励语言设计者的创新，但也会让开发者处于一个不利的状况：他们会为了支持不同的实现和运行环境而写一些代码（polyfills）[1]。

ECMAScript 的规范会因为各种不同原因（原因太多，不一一列举）不断变化。根本上讲，这些变化是为了使用新的方法来解决老的问题，或者用于支持在巨型计算生态系统中的改进。而不断变化的规范就形成了语言的逐步进化。因此，虽然我要讲的是“核心概念”，它们听起来是不变的，但其实并非如此。本章探讨的概念是那些最基本和重要的，但不要忘记，时刻要准备应对新变化的到来。

脚本设计

正如其名称所示，ECMAScript 是脚本语言，使用程序化的方式与主机环境进行交互。无论是浏览器、服务器或单片机，都可以为 JavaScript 暴露一些可操作的接口。大多数宿主环境仅允许 JavaScript 去触发那些本来用户就可以操作的功能（虽然用户是手动的）。例如，在浏览器中，用户可以使用鼠标或手指在网页链接上单击，JavaScript 则可以用程序的形式完成单击的操作，如下所示：

```
document.getElementById('search').click();
```

从传统观念上讲，ECMAScript 几乎专门被设计成一种工具，用于在浏览器中编写网络脚本。开发人员用它来提升浏览网页时的用户体验。如今，ECMAScript 能用于服务器上，就像它被应用到浏览器中一样，这一切归功于 V8 或 TraceMonkey 这样的独立引擎。

ECMAScript 标准组织预见到开发人员使用 JavaScript 的传统方式和它目前成长的领域非常不一样。在最近的规范中它聪明地界定了什么是“网络脚本”，同时列举了两个当下比较流行的应用情境。

Web 浏览器为 ECMAScript 在客户端的运行提供了宿主环境，它封装了一系列对象供 ECMAScript 使用，包括窗口、菜单、弹窗、对话框、文本区域、锚点、多窗口的页面、历史、cookie 和输入/输出对象。此外，宿主环境提供了一种方式，将脚本代码附着在事件上，例如焦点的改变，页面和图像的加载、卸载，错误和中断，选择，表单提交和鼠标操作。含有脚本代码的 HTML 和被显示的页面是一种组合，脚本代码提供用户接口，而显示页面则提供静态或动态的文字和图片。脚本代码用于响应用户交互，也就不必要有个主程序了。

Web 服务器为服务端的计算提供了不同的宿主环境，包括代表请求、客户端和文件的对象，还包括锁定以及共享数据的机制。在浏览器和服务端同时使用脚本语言，使得将计算逻辑分布到客户端和服务端的同时能为基于 Web 的应用提供定制化的用户接口。

每个支持 ECMAScript 的 Web 浏览器和服务器，都需要有满足 ECMAScript 运行的宿主环境。

[1] http://remysharp.com/2010/10/08/what-is-a-polyfill/。

注 在作者写作本书时，最新版本的 ECMAScript 6（命名为“Harmony”）已经快要发布了，虽然还没有官方发布，但许多提案修改已经被一些运行时引擎和浏览器支持了。本章会对语言的核心进行详尽说明，其中也包括一些在“Harmony”中刚刚引入的新特性。如果某个特性还没有被大范围支持，我会特别提醒读者注意。

1.2 对象概述

JavaScript 是由 Brendan Eich 创建的一种面向对象编程（OOP）语言，当时他还在 Netscape 公司工作，花了几周的开发时间就发布了。虽然 JavaScript 的名字中有个“Java”，但它实际上跟 Java 语言没什么关系。在 InfoWorld 的一篇对 Eich 的采访稿中，他解释了 JavaScript 命名的由来：

InfoWorld：据我所知，JavaScript 开始的时候叫 Mocha，后来改名叫 LiveScript，在 Netscape 和 Sun 合并以后才叫 JavaScript 的。但实际上它跟 Java 没什么关系，是这样吗？

Eich：没错。从 Mocha 变到 LiveScript，是从 5 月到 12 月（1995）的事情。后来到了 12 月初，Netscape 跟 Sun 签了一个许可协议，它就改名为 JavaScript 了。而当时的想法是，让 JavaScript 成为补充编译型语言 Java 的一种脚本语言[1]。

即使随便比较一下这两种语言都能看出，其实它们是完全不一样的。跟 Java 很不同的一点是，JavaScript 不需要编译，非强类型，也没有一种正规的基于类的继承机制。相反，JavaScript 运行在宿主环境的上下文中（比如一个 Web 浏览器），支持动态类型的变量，通过原型链而不是类来实现继承。因此，我们可以认为当时应该是想让人们因为 JavaScript 与 Java 名字相似而记住它，并对市场宣传有一定的帮助，其实这两种语言没有什么实质上的联系。

然而，尽管它们大相径庭，但 Java 和 JavaScript 都是面向对象编程语言家族中的一员。面向对象是对象通过相互之间的通信而控制程序执行的方法。它是一种比较流行的编程范式，除此之外还有函数式（Functional）、命令式（Imperative）和声明式（Declarative）。

注 JavaScript 虽然被大家普遍认为是面向对象的编程语言，但并不意味着它不支持其他的编程范式。例如，流行类库 Underscore.js[2]就是用函数式的风格编写而成的。

1.2.1 对象化

作为一个面向对象的编程语言意味着什么呢？对一个有经验的程序员来说这可能不是个问题，但是回答这个问题会给你机会来评价一下 JavaScript 实现面向对象的方式。本书大量的篇幅会让你设计和思考对象以及它们之间的关系。但要记住，对象只是用于编程建模的多种隐喻（metaphors）之一。

隐喻通常都诱人而又晦涩难懂。对它的理解可以让你更加清晰地构思一个问题的解决方案，

[1] http://www.infoworld.com/d/developer-world/javascript-creator-ponders-past-future-704。

[2] http://underscorejs.org/。

而不是陷入不必要的复杂的泥潭。当你要回答面向对象编程意味着什么时，思考下你自己的理解和预设，你可能会发现自己对这个问题的理解是有偏见的。

在 JavaScript 中，对象仅仅是属性（properties）的容器。我听说有的程序员把对象形容成“属性包”，听起来很有画面感。每个对象都可以有零个或多个属性，这些属性可以持有一个基本类型（primitive）的值，也可以指向一个复杂类型的对象。JavaScript 可以通过三种方式创建对象：使用字面量（literal notation）、`new()`运算符或 `create()`函数。这三种方式可以简单表示如下：

```
var foo = {},
    bar = new Object(),
    baz = Object.create(null);
```

这些方法之间的区别在于对象初始化的方式，稍后我们会详细解释。现在让我来看一下，如何通过自定义属性来修饰对象。

1. 属性管理器

很多开发人员都认为对象的属性只是一个容器，用于将 name 和 value 赋进去。实际上，JavaScript 让开发人员可以通过一系列强大的属性描述符，进一步定制属性的行为。下面我们逐个分析一下。

（1）可配置特性（configurable）

当这个特性（attribute）设为 true 时，属性可以从父对象中删除，未来还可以修改属性的描述符；当设置为 false 时，属性的描述符会被锁定，无法修改。下面是一个简单的例子：

```
var car = {};

// A car can have any number of doors
Object.defineProperty(car, 'doors', {
    configurable: true,
    value: 4
});

// A car must have only four wheels
Object.defineProperty(car, 'wheels', {
    configurable: false,
    value: 4
});

delete car.doors;

// => "undefined"
console.log(car.doors);

delete car.wheels;
// => "4"
```

```
console.log(car.wheels);
Object.defineProperty(car, 'doors', {
    configurable: true,
    value: 5
});

// => "5"
console.log(car.doors);

// => Uncaught TypeError: Cannot redefine property: wheels
Object.defineProperty(car, 'wheels', {
    configurable: true,
    value: 4
});
```

正如在例子中看到的，wheel 属性不可变而 doors 属性仍然可变。程序员可能会将属性的 configurable 特性设为 false，用于保护对象不被修改，这是一种防御性编程的形式，就像语言中内置对象一样。

（2）可枚举特性（enumberable）

如果对象的属性可以使用代码来遍历，那这些属性就是可枚举（enumberable）的。当将其设为 false 时，这些属性就不能被遍历了。举个例子：

```
var car = {};

Object.defineProperty(car, 'doors', {
  writable: true,
  configurable: true,
  enumerable: true,
  value: 4
});

Object.defineProperty(car, 'wheels', {
  writable: true,
  configurable: true,
  enumerable: true,
  value: 4
});

Object.defineProperty(car, 'secretTrackingDeviceEnabled', {
  enumerable: false,
  value: true
});

// => doors
// => wheels
```

```
for (var x in car) {
  console.log(x);
}
// => ["doors", "wheels"]
console.log(Object.keys(car));

// => ["doors", "wheels", "secretTrackingDeviceEnabled"]
console.log(Object.getOwnPropertyNames(car));

// => false
console.log(car.propertyIsEnumerable('secretTrackingDeviceEnabled'));

// => true
console.log(car.secretTrackingDeviceEnabled);
```

正如在例子中看到的，即使一个属性不是 enumerable 的，这也不意味着这个属性是完全被隐藏起来的。enumerable 特性可以阻止程序员使用某些属性，但它不应该作为一种属性不被检视的方式。

（3）可写特性（writable）

当可写特性（writable）为 true 时，与属性相关联的值是可以改变的；否则，值不可改变。

```
var car = {};

Object.defineProperty(car, 'wheels', {
  value: 4,
  writable: false
});

// => 4
console.log(car.wheels);

car.wheels = 5;

// => 4
console.log(car.wheels);
```

2. 检视对象

在上一小节中，你知道了如何定义对象的属性。在这一节中，你会学到如何在 JavaScript 中深入地挖掘对象，这和生活中知道如何读写一样有用。下面是一系列检视对象时需要了解的函数和属性。

（1）Object.getOwnPropertyDescriptor

在上一小节中，你看到了很多用于设置属性特性的方式。`Object.getOwnPropertyDescriptor` 可以详细告诉你对象属性特性的配置。

```
var o = {foo : 'bar'};

// Object {value: "bar", writable: true, enumerable: true, configurable: true}
Object.getOwnPropertyDescriptor(o,'foo');
```

（2）Object.getOwnPropertyNames

这个方法返回对象全部属性的名字，包括那些不能枚举的：

```
var box = Object.create({}, {
    openLid: {
        value: function () {
            return "nothing";
        },
        enumerable: true
    },
    openSecretCompartment: {
        value: function () {
            return 'treasure';
        },
        enumerable: false
    }
});

// => ["openLid", "openSecretCompartment"]
console.log(Object.getOwnPropertyNames(box).sort());
```

（3）Object.getPrototypeOf

这个方法用来返回特定对象的原型。有时还可以使用__proto__方法来代替这个方法，很多解释器的实现会使用这个方法来获取对象的原型。然而，使用__proto__总让人感觉有点 hack 的味道，JavaScript 社区也主要把它用作权宜之计。然而值得注意的是，即使 Object.getPrototypeOf 可以让你访问一个对象的原型，但是设置一个对象实例的原型的唯一方法是使用__proto__属性。

```
var a = {};

// => true
console.log(Object.getPrototypeOf(a) === Object.prototype && Object.prototype === a.__proto__);
```

（4）Object.hasOwnProperty

JavaScript 的原型链可以让你通过遍历一个对象的实例，返回所有可枚举的属性，包括不存在于这个对象上但存在于原型链中的属性。hasOwnProperty 方法可以让你分辨出某个属性是否存在于对象实例中：

```
var foo = {
    foo: 'foo'
};
var bar = Object.create(foo, {
    bar: {
```

```
        enumerable: true,
        value: 'bar'
    }
});
// => bar
// => foo
for (var x in bar) {
    console.log(x);
}

var myProps = Object.getOwnPropertyNames(bar).map(function (i) {
    return bar.hasOwnProperty(i) ? i : undefined;
});

// => ['bar']
console.log(myProps);
```

（5）Object.keys

这个方法仅返回对象中可枚举的属性：

```
var box = Object.create({}, {
    openLid: {
        value: function () {
            return "nothing";
        },
        enumerable: true
    },
    openSecretCompartment: {
        value: function () {
            return 'treasure';
        },
        enumerable: false
    }
});

// => ["openLid"]
console.log(Object.keys(box));
```

（6）Object.isFrozen

如果对象不能扩展，属性也不能修改，那么这个方法返回 true，反之返回 false。

```
var bombPop = {
    wrapping: 'plastic',
    flavors: ['Cherry', 'Lime', 'Blue Raspberry']
};

// => false
console.log(Object.isFrozen(bombPop));
```

```
delete bombPop.wrapping;
// undefined;
console.log(bombPop.wrapping);

// prevent further modifications
Object.freeze(bombPop);

delete bombPop.flavors;

// => ["Cherry", "Lime", "Blue Raspberry"]
console.log(bombPop.flavors);

// => true
console.log(Object.isFrozen(bombPop));
```

（7）Object.isPrototypeOf

这个方法在对象的整个原型链中检查每一环，看传入的对象是否存在于其中：

```
// => true
Object.prototype.isPrototypeOf([]);

// => true
Function.prototype.isPrototypeOf(()=>{});

// => true
Function.prototype.isPrototypeOf(function(){});

// => true
Object.prototype.isPrototypeOf(()=>{});
```

注 在写这篇文章的时候，=>箭头语法仅被如 Firefox 22（SpiderMonkey 22）等浏览器支持。这种语法在不支持的浏览器中运行会产生语法错误。

（8）Object.isExtensible

默认情况下，在 JavaScript 中新生成的对象是可扩展的，即可添加新的属性。然而，对象在未来可以被标记为不可扩展。某些环境下，在一个不可扩展的对象上设置属性会抛出错误。在试图修改一个对象前，你可以使用 `Object.isExtensible` 来检查这个对象是否可被修改：

```
var car = {
    doors: 4
};

// => true
console.log(Object.isExtensible(car) === true);

Object.preventExtensions(car);

// => false
```