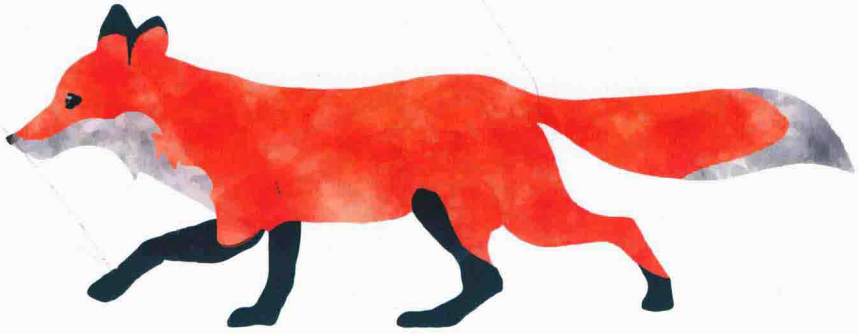
Broadview®
www.broadview.com.cn

本书并没有把所有的前端知识按部就班地罗列出来，而是努力地向读者传递一种行之有效的学习方法，总结出了一套适合大多数人的学习方式：围绕核心，渐进增强。

JavaScript
核心技术开发解密

阳波 编著



中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

JavaScript
核心技术开发解密

阳波 编著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内容简介

本书针对 JavaScript 中的核心技术，结合前沿开发实践，对 JavaScript 的内存、函数、执行上下文、闭包、面向对象、模块等重点知识，进行系统全面的讲解与分析。每一个知识点都以实际应用为依托，帮助读者更加直观地吸收知识点，为学习目前行业里的流行框架打下坚实基础。本书适合 JavaScript 初学者，有一定开发经验但是对于 JavaScript 了解不够的读者，以及开发经验丰富但没有形成自己知识体系的前端从业者。

图书在版编目（CIP）数据

JavaScript 核心技术开发解密 / 阳波编著. – 北京: 电子工业出版社, 2018.3

ISBN 978-7-121-33696-6

I. ①J… II. ①阳… III. ①JAVA 语言—程序设计 IV. ①TP312

中国版本图书馆 CIP 数据核字（2018）第 029430 号

策划编辑：安　娜
责任编辑：安　娜
印　　刷：三河市华成印务有限公司
装　　订：三河市华成印务有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：787×980　1/16　　印张：14.5　字数：276 千字
版　　次：2018 年 3 月第 1 版
印　　次：2018 年 3 月第 1 次印刷
定　　价：69.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819，faq@phei.com.cn。

前言

在阅读这本书之前，不知道大家有没有思考过一个问题：

前端学习到底有没有捷径？

在我看来，学习的捷径并不是指不用付出多少努力就能够获得成功，而是在我们付出努力之后，能够感觉到自己的努力没有白费，能够学到更多的知识，能够真正做到一分耕耘，一分收获。

所以学习有没有捷径？我的答案是：一定有。

其实大多数人并不是不想付出努力，而是不知道如何去努力，不知道如何有效地努力。我们想要学好一个知识，想要掌握一门技术，但是往往不知从何下手。

前端的学习也是如此。也许上手简单的 HTML/CSS 知识，会给刚开始学习的读者一个掌握起来很容易的印象。但是整个前端知识体系繁杂而庞大，导致大多数人在掌握了一些知识之后，仍然觉得自己并没有真正入门，特别是近几年，前端行业的从业人员所要掌握的知识越来越庞杂，入门的门槛也越来越高，甚至进阶道路也变成了一场马拉松。

也许在几年以前，我们只需会用 jQuery 就可以说自己是一名合格的前端开发者，但是现在的 JavaScript 语言已经不再是几年前那样，只需处理一些简单的逻辑就足够了。随着前后端分离的方式被越来越多的团队运用于实践，用户对 UI 的要求越来越高，对性能的要求也越来越高，JavaScript 承载了更多的任务。虽然前端行业大热，但是我们的学习压力也随之倍增。

所以我一直在思考，在这样的大环境背景下，对于新人朋友来说，什么样的学习方式能让我们的学习效率更高？或者说，一本什么样的前端书籍才算是好书？

是将所有的前端知识按部就班地罗列出来告诉大家吗？肯定不是。

很多书籍，以及各类官方文档其实都在做这件事。但是对于多数读者来说，把所有知识罗列出来摆在眼前，并不是一个能够掌握它们的有效方式。这种学习方法不仅效率低下，而且学完之后，也并不知道在实践中到底如何使用它们，我们其实是迷茫的。

所以，如果有一本书，它在努力地向读者传递一种行之有效的学习方法，那么对于适合这种学习方法的读者来说，就一定是一本好书。

这就是我们这本书努力的方向。

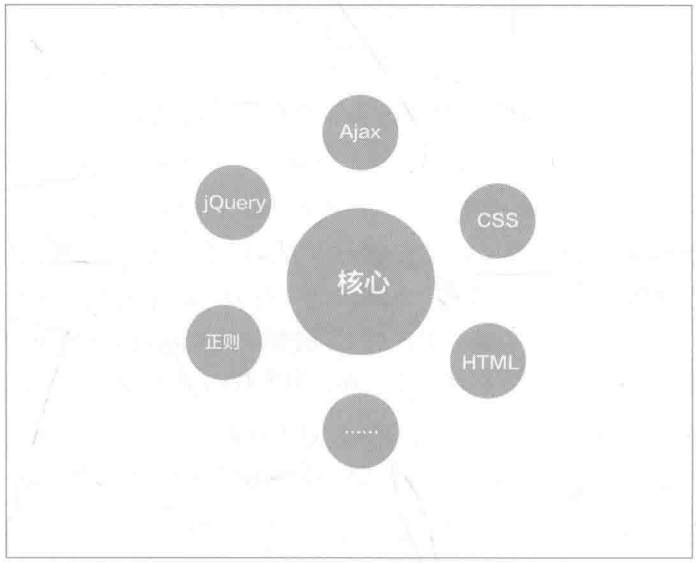
凭借多年的工作经验，在长期写博客并与读者互动的过程中，我总结出了一套适合大多数人的学习方式，那就是：

围绕核心，渐进增强。

本书将整个 JavaScript 相关的知识点简单粗暴地划分为核心知识与周边知识。

周边知识的特点就是相对独立，不用非得学会了其他的知识点之后才能掌握它，也不用掌握了它之后才能学习其他的知识。例如 Ajax，如果仅仅只是想要使用它，那么用别人封装好的方法，通过官方文档或者搜索引擎，只需要两分钟你就知道怎么使用。周边知识不会成为我们学习的瓶颈。

而核心知识不一样，核心知识是整个前端知识体系的骨架所在。它们前后依赖，环环相扣。例如，在核心知识链中，如果你搞不清楚内存空间管理，你可能就不会真正明白闭包的原理，就没办法完全理解原型链，这是一个知识的递进过程。我们在学习过程中遇到的瓶颈，往往都是由于某一个环节的核心知识没有完整掌握造成的。而核心知识的另一个重要性就在于，它们能够帮助我们更加轻松地掌握其余的周边知识。



所以，如果读者知道这条核心知识链到底是什么，并且彻底地掌握它们，那么你就已经具备了成为一名优秀前端程序员的能力。这样的能力能够让你在学习其他知识点的时候方向明确，并且充满底气。

所以这本书的主要目的就在于帮助读者拥有这样的进阶能力。

基于这个思路，这本书的呈现方式必定与其他图书不同。本书不会按部就班地告诉你如何声

明变量、如何声明函数，不会罗列出所有的基础知识，对于基础知识的传授，《JavaScript 高级编程》已经做得足够好，因此没有必要重复做同样的事情。我会一步一步与大家分享这条完整的核心链。我的期望是，当大家学完这本书中的知识后，能够对前端开发的现状有一个大致的了解，知道什么知识是最有用的，什么知识是工作中需要的，拥有进一步学习流行前端框架的能力，拥有在前端方向自主学习、自主进步的知识基础与能力。

最后希望在这本书的陪伴下，大家能有一个愉快的、充实的学习历程。

读者服务

轻松注册成为博文视点社区用户（www.broadview.com.cn），扫码直达本书页面。

- **下载资源**：本书如提供示例代码及资源文件，均可在 下载资源 处下载。
- **提交勘误**：您对书中内容的修改意见可在 提交勘误 处提交，若被采纳，将获赠博文视点社区积分（在您购买电子书时，积分可用来抵扣相应金额）。
- **交流互动**：在页面下方 读者评论 处留下您的疑问或观点，与我们和其他读者一同学习交流。

页面入口：http://www.broadview.com.cn/33696

目录

1 三种基础数据结构

在 JavaScript 中，有三种常用的数据结构是我们必须了解的，它们分别是栈（stack）、堆（heap）、队列（queue）。它们是理解整个核心的基础，在 JavaScript 中分别有不同的应用场景，因此先来介绍它们。

1.1 栈

当我们在学习中遇到栈这个名词时，可能面临的是不同的含义。如果没有理清不同的应用场景，就会给我们的理解带来困惑。

场景 1：栈是一种数据结构，它表达的是数据的一种存取方式，这是一种理论基础。

场景 2：栈可用来规定代码的执行顺序，在 JavaScript 中叫作函数调用栈（call stack），它是根据栈数据结构理论而实现的一种实践。理解函数调用栈的概念非常重要，我们会在后续的章节里详细说明。

场景 3：栈表达的是一种数据在内存中的存储区域，通常叫作栈区。但是 JavaScript 作为一门高级程序语言，并没有同其他语言那样区分栈区或堆区，因此这里不做扩展。我们可以简单粗暴地认为在 JavaScript 中，所有的数据都是存放在堆内存空间中的。

这里需要重点掌握栈这种数据结构的原理与特点，学习它的最终目的是掌握函数调用栈的运行方式。下面可以通过乒乓球盒子这个案例来简单理解栈的存取方式，如图1-1所示。

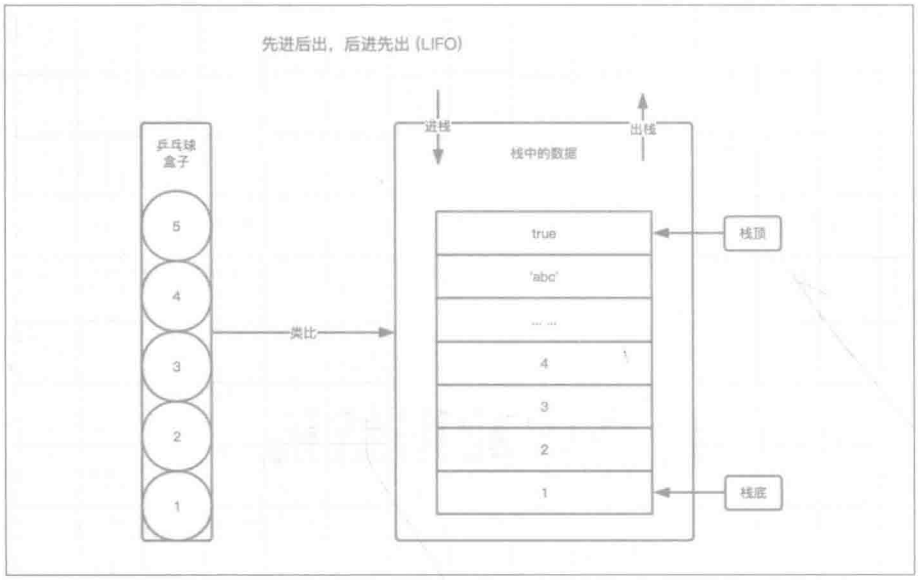


图 1-1　乒乓球盒子

往图1-1所示的乒乓球盒子中依次放入乒乓球，当想要取出来使用的时候，处于盒子顶层的乒乓球 5，它一定是最后被放进去并且最先被取出来的。而要想使用底层的乒乓球 1，则必须先将上面的所有乒乓球取出来之后才能取出。但乒乓球 1 是最先放入盒子的。

这种乒乓球的存取方式与栈数据结构如出一辙。这种存取方式的特点可总结为**先进后出，后进先出**（LIFO，Last In，First Out）。如图1-1右侧所示，处于栈顶的数据 true，最后进栈，最先出栈。处于栈底的数据 1，最先进栈，最后出栈。

在 JavaScript 中，数组（Array）提供了两个栈方法来对应栈的这种存取方式，它们在实践中十分常用。

push：向数组末尾添加元素（进栈方法）。

push 方法可以接收任意参数，并把它们逐个添加到数组末尾，并返回数组修改后的长度。

```
var a = [];
a.push(1);          // a: [1]
a.push(2, 4, 6);    // a: [1, 2, 4, 6]
var l = a.push(5); // a: [1, 2, 4, 6, 5] l: 5
```

pop：弹出数据最末尾的一个元素（出栈方法）。

pop 方法会删除数组最末尾的一个元素，并返回。

```
var a = [1, 2, 3];
a.pop(); // a: [1, 2]

// a.pop()的返回结果为 3
```

1.2 堆

堆数据结构通常是一种树状结构。

它的存取方式与在书架中取书的方式非常相似。书虽然整齐地摆放在书架上，但是只要知道书的名字，在书架中找到它之后就可以很方便地取出，我们甚至不用关心书的存放顺序，即不用像从乒乓球盒子中取乒乓球那样，必须将一些乒乓球拿出来之后才能取到中间的某一个乒乓球。

图 1-2是 testHeap 示意图。

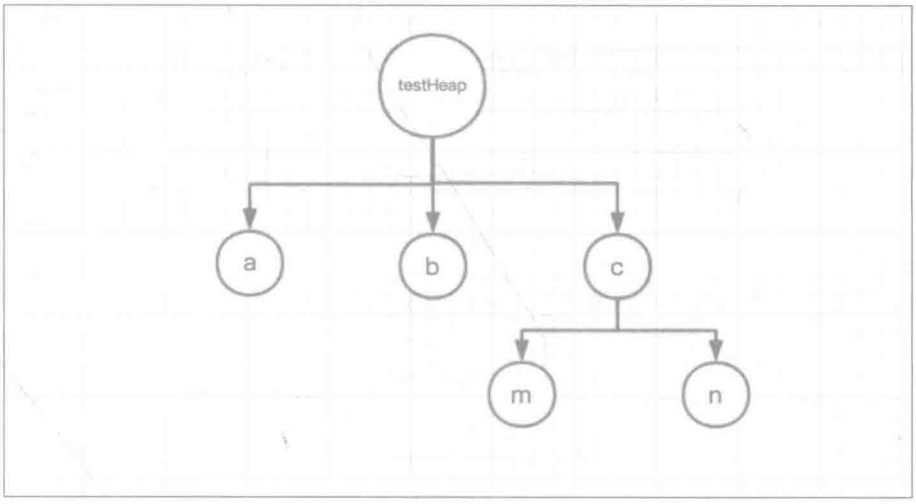


图 1-2 testHeap 示意图

该示意图可以用字面量对象的形式体现出来。

```
var testHeap = {
    a: 10,
    b: 20,
```

```
    c: {
        m: 100,
        n: 110
    }
}
```

当我们想要访问 a 时，只需通过 testHeap.a 来访问即可，而不用关心 a、b、c 的具体顺序。

1.3 队列

在 JavaScript 中，理解队列数据结构的目的是为了搞清楚事件循环（Event Loop）机制到底是怎么回事。在后续的章节中会详细分析事件循环机制。

队列（queue）是一种先进先出（FIFO）的数据结构。正如排队过安检一样，排在队伍前面的人一定是最先过安检的人。队列原理如图 1-3所示。

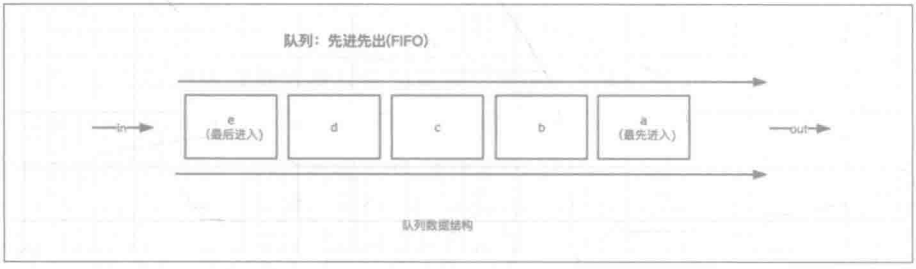


图 1-3 队列原理

2 内存空间

因为 JavaScript 有垃圾自动回收机制，所以对于前端开发人员来说，内存空间并不是一个经常被提及的概念，很容易被大家忽视。特别是很多非计算机专业的读者在进入前端行业之后，通常对内存空间的认知比较模糊，甚至一无所知。但是内存空间却是真正的基础，这是我们进一步理解闭包等重要概念的理论基石，所以非常有必要花费一点时间去了解它。

2.1 基础数据类型与变量对象

最新的 ECMAScript 标准号定义了 7 种数据类型，其中包括六种基础数据类型与一种引用数据类型（Object）。

其中基础数据类型表如表 2-1 所示。

表 2-1 基础数据类型表

类 型	值
Boolean	只有两个值：true 与 false
Null	只有一个值：null
Undefined	只有一个值：undefined
Number	所有的数字
String	所有的字符串
Symbol	符号类型 var sym = Symbol('testsymbol')

由于目前常用的浏览器版本还不支持 Symbol，而且通过 babel 编译之后的代码量过大，因此在实践中建议暂时不要使用 Symbol。

下面来探讨一个问题，有一个很简单的例子如下所示。

```
function fn() {
  var a1 = 10;
  var a2 = 'hello';
  var a3 = null;
}
```

现在需要思考的是，当运行函数 fn 时，它其中的变量 a1、a2、a3 都保存在什么地方？

函数运行时，会创建一个执行环境，这个执行环境叫作执行上下文（Execution Context，我们会在后续的章节详细介绍它）。在执行上下文中，会创建一个叫作变量对象（VO，后续章节详细学习）的特殊对象。基础数据类型往往都保存在变量对象中，如图 2-1所示。

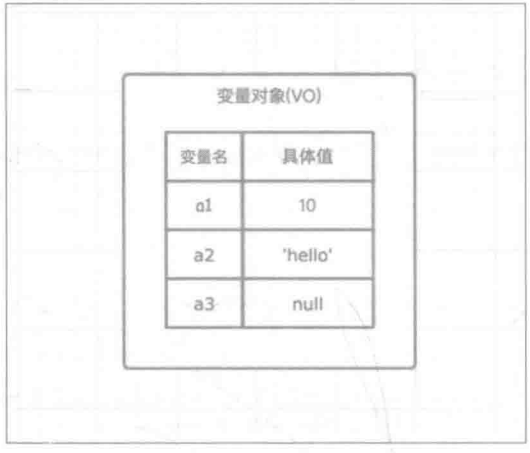


图 2-1　变量对象

变量对象也存在于堆内存中，但是由于变量对象有特殊职能，因此在理解时，建议仍然将其与堆内存空间区分开来。

2.2 引用数据类型与堆内存空间

引用数据类型（Object）的值是保存在堆内存空间中的对象。在 JavaScript 中，不允许直接访问堆内存空间中的数据，因此不能直接操作对象的堆内存空间。在操作对象时，实际上是在操作对象的引用而不是实际的对象。因此，引用数据类型都是按引用访问的。这里的引用，可以理解为保存在变量对象中的一个地址，该地址与堆内存中的对象相关联。

为了更好地理解变量对象与堆内存，下面用一个例子与图解配合讲解。

```
function foo() {
    var a1 = 10;
    var a2 = 'hello';
    var a3 = null;
    var b = { m: 20 };
    var c = [1, 2, 3];
}
```

如图 2-2所示，当我们想要访问堆内存空间中的引用数据类型时，实际上是通过一个引用（地址指针）来访问的。

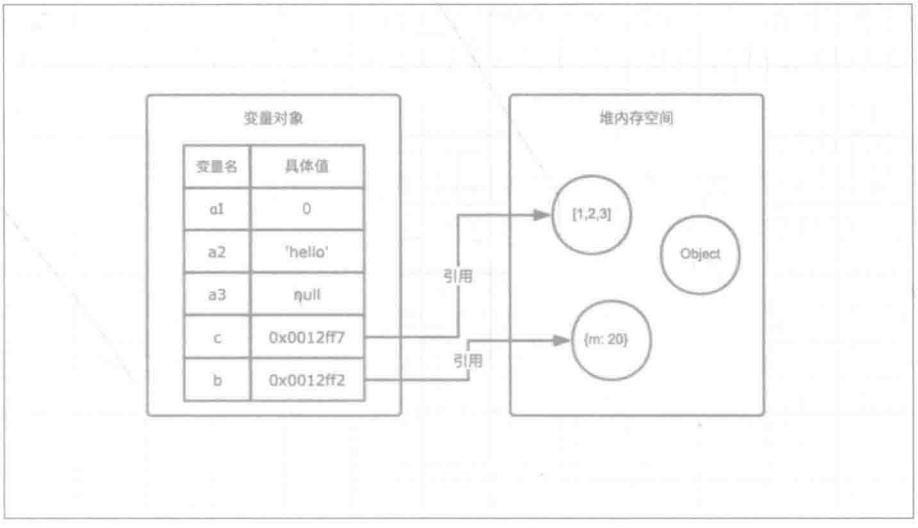


图 2-2 变量对象地址指针

在前端面试题中，我们常常会遇到这样一个类似的题目。

```
// demo01.js
var a = 20;
var b = a;
b = 30;

// 这时a的值是多少
```

```
// demo02.js
var m = { a: 10, b: 20 }
var n = m;
n.a = 15;

// 这时m.a的值是多少
```

在 demo01 中，基础数据类型发生了一次复制行为。在 demo02 中，引用数据类型发生了一次复制行为。

当变量对象中的数据发生复制行为时，新的变量会被分配到一个新的值。在 demo01 中，通过 var b = a 发生复制之后，虽然 *a* 与 *b* 的值都等于 20，但它们其实已经是相互独立互不影响的值了。因此当我们修改了 *b* 的值以后，*a* 的值并不会发生变化。具体如图 2-3所示。

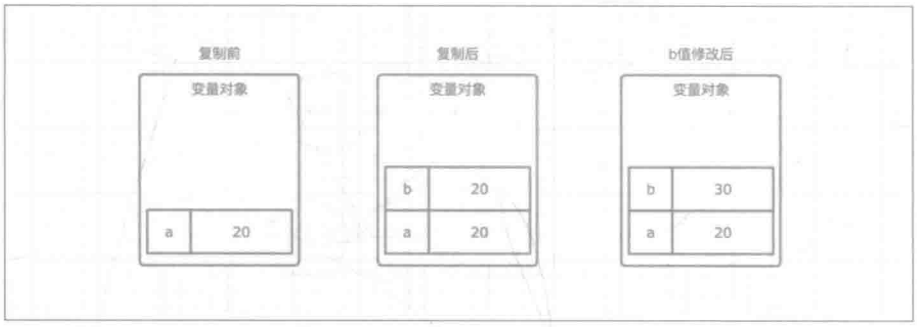


图 2-3　变量对象复制

在 demo02 中，通过 var n = m 发生了一次复制行为。引用类型的复制同样会为新的变量自动分配一个新的值并保存在变量对象中。但不同的是，这个新的值，仅仅只是引用类型的一个地址指针。当地址指针相同时，尽管它们相互独立，但是它们指向的具体对象实际上是同一个。