O'REILLY®

TURING 图灵程序设计丛书

Head First JavaScript 程序设计

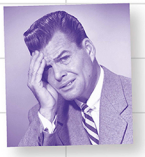


提防常见的
JavaScript陷阱

JavaScript编程
学习指南

迅速开启职业
编程生涯





避免难堪的
类型转换错误



领悟朋友有关函数和
对象的每个说法
都可能不对的原因



沉醉于120个
填字游戏和练习



[美] Eric T. Freeman Elisabeth Robson 著
袁国忠 译

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

数字版权声明

图灵社区的电子书没有采用专有客户端，您可以在任意设备上，用自己喜欢的浏览器和PDF阅读器进行阅读。

但您购买的电子书仅供您个人使用，未经授权，不得进行传播。

我们愿意相信读者具有这样的良知和觉悟，与我们共同保护知识产权。

如果购买者有侵权行为，我们可能对该用户实施包括但不限于关闭该帐号等维权措施，并可能追究法律责任。

TURING 图灵程序设计丛书

Head First JavaScript 程序设计



[美] Eric T. Freeman
Elisabeth Robson 著
袁国忠 译

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

O'Reilly Media, Inc. 授权人民邮电出版社出版

人民邮电出版社
北 京

图书在版编目（CIP）数据

Head First JavaScript 程序设计 /（美）埃里克·T. 弗里曼（Eric T. Freeman），（美）伊丽莎白·罗布森（Elisabeth Robson）著；袁国忠译 . —北京：人民邮电出版社，2017.8

（图灵程序设计丛书）

书名原文：Head First JavaScript Programming

ISBN 978-7-115-45841-4

I. ① H… II. ①埃… ②伊… ③袁… III. ① JAVA 语言－程序设计 IV. ① TP312.8

中国版本图书馆 CIP 数据核字 (2017) 第 119745 号

内 容 提 要

本书语言和版式活泼，内容讲解深入浅出，是难得的JavaScript入门书。本书内容涵盖JavaScript的基本知识以及对象、函数和浏览器文档对象模型等高阶主题。书中配备了大量有趣的示例、图示和练习，让读者轻轻松松掌握JavaScript编程。

本书的读者对象为JavaScript入门读者以及网页设计入门者。

♦ 著 ［美］Eric T. Freeman Elisabeth Robson
译 袁国忠
责任编辑 杨 琳
责任印制 彭志环

♦ 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 印刷

♦ 开本：880 × 1230 1/20
印张：34.8
字数：1028 千字 2017 年 8 月第 1 版
印数：1－4 000 册 2017 年 8 月北京第 1 次印刷
著作权合同登记号 图字：01-2015-5420 号

定价：129.00 元

读者服务热线：(010) 51095186 转 600 印装质量热线：(010) 81055316
反盗版热线：(010) 81055315
广告经营许可证：京东工商广登字 20170147 号

致JavaScript：你出身并不高贵，却轻松击败了所有在浏览器领域向你发起挑战的语言。

O’Reilly Media, Inc. 介绍

O’Reilly Media 通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自 1978 年开始，O’Reilly 一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O’Reilly 的发展充满了对创新的倡导、创造和发扬光大。

O’Reilly 为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了 *Make* 杂志，从而成为 DIY 革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O’Reilly 的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O’Reilly 现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版，在线服务或者面授课程，每一项 O’Reilly 的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O’Reilly Radar 博客有口皆碑。”

——*Wired*

“O’Reilly 凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——*Business 2.0*

“O’Reilly Conference 是聚集关键思想领袖的绝对典范。”

——*CRN*

“一本 O’Reilly 的书就代表一个有用、有前途、需要学习的主题。”

——*Irish Times*

“Tim 是位特立独行的商人，他不光放眼于最长远、最广阔的视野并且切实地按照 Yogi Berra 的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去 Tim 似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——*Linux Journal*

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

对本书的赞誉

“警告：本书寓教于乐，你若想苦哈哈地学习，那千万不要选这本书。它还有个副作用，你学到的JavaScript知识比看一般的技术书要多。”

——**Jesse Palmer，Gannett Digital资深软件开发人员**

“如果每个中小学生都有机会阅读Elisabeth和Eric编著的《Head First HTML与CSS》，且本书和*Head First HTML5 Programming*也得以纳入高中数学和科学课程，那么我们国家就永远不会丧失竞争优势了。”

——**Michael Murphy，The History Tree资深系统咨询师**

“Head First系列丛书充分利用了建构主义等现代学习理论，帮助读者快速掌握新知识。本书表明，即便是专家级内容，也可高效而快速地传授给读者。别误会，这可是一部严肃的JavaScript著作，但让人读起来趣味盎然！”

——**Frank Moore，Web设计师和开发人员**

“想找一部读起来令人兴趣盎然、笑声不断，又能学到编程技能的著作吗？本书就是这样的著作！”

——**Tim Williams，软件领域创业者**

“不管你的编程水平如何，都请将本书纳入你的书单中！”

——**Chris Fuselier，工程咨询师**

“Robson和Freeman再次获得了成功！本书秉承了他们编著的其他Head First系列丛书内容丰富而有趣的写作风格，引导读者完成一些有趣而实用的项目，为使用现代JavaScript编程技巧解决实际问题打下了坚实的基础。”

—— **Russell Alleen-Willems，DiachronicDesign.com数字考古学家**

“Freeman和Robson一如既往地采用创新性的教学手法，向读者传授基本原理和复杂概念。”

——**Mark Arana，华特迪士尼公司战略与创新部**

对Eric T. Freeman和Elisabeth Robson的其他著作的赞誉

“这本书很对心无旁骛、桀骜不羁的编码大师的口味。很不错的实用开发策略指南，没有又长又臭的陈词滥调，能迅速让我的大脑活跃起来。”

——**Travis Kalanick，优步CEO**

“本书语言清晰幽默，内容充实，即便读者不是程序员，也能从中学到解决问题的思路。”

——**Cory Doctorow，Boing Boing联合编辑和科幻小说作家**

“学海无涯，举重若轻，何需汗牛充栋。”

——**Ward Cunningham，Wiki之父**

“在我阅读过的著作中，让我觉得不可错过的如凤毛麟角（最多不超过10本），本书位列其中。”

——**David Gelernter，耶鲁大学计算机教授**

“它深深地打动了我，让我又哭又笑。”

——**Daniel Steinberg，java.net主编**

“我想不出还有哪位导游比Eric和Elisabeth更优秀了。”

——**Miko Matsumura，Hazelcast市场营销和开发人员关系副总裁（Sun公司前首席Java布道师）**

“本书让我爱不释手，忍不住当着妻子的面亲吻它。”

——**Satish Kumar**

“运用大量图表，循序渐进，准确模拟了学习JavaScript的最佳方式。”

——**Danny Goodman，*Dynamic HTML: The Definitive Guide*作者**

“Eric和Elisabeth对其要介绍的内容了如指掌。随着Internet越来越复杂，在制作网页的过程中，灵感显得日益重要。每一章都精心设计，既措辞巧妙又不失实用地讲解了每个概念。”

——**Ken Goldstein，Shop.com前CEO和*This is Rage: A Novel of Silicon Valley and Other Madness*作者**

Eric T. Freeman和Elisabeth Robson的其他O'Reilly图书

Head First Design Patterns

Head First HTML and CSS

Head First HTML5 Programming

O'Reilly的其他相关图书

Head First HTML5 Programming

JavaScript: The Definitive Guide

JavaScript Enlightenment

O'Reilly Head First系列的其他图书

Head First HTML and CSS

Head First HTML5 Programming

Head First Design Patterns

Head First Servlets and JSP

Head First SQL

Head First Software Development

Head First C#

Head First Java

Head First Object-Oriented Analysis and Design (OOA&D)

Head First Ajax

Head First Rails

Head First PHP & MySQL

Head First Web Design

Head First Networking

Head First iPhone and iPad Development

Head First jQuery

作者简介



Eric Freeman



Elisabeth Robson

Eric Freeman 做过时髦的黑客、公司副总裁、工程师、智囊等，谙熟多个领域的语言、惯例和文化，Head First系列丛书创立者之一Kathy Sierra视其为难得的多面手。

在职业生涯方面，Eric最近辞去了担任了近10年之久的媒体公司高管职位——华特迪士尼公司DisneyOnline & Disney.com的首席技术官，将全部精力都投入到他与Elisabeth创建的创业公司WickedlySmart。

在教育背景方面，Eric是一位计算机科学家，在耶鲁大学攻读博士学位期间，一直与行业翘楚David Gelernter一起从事研究工作，其博士论文首次实现了他与Gelernter博士提出的活动流概念，对桌面替代品的探索工作产生了深远的影响。

闲暇之余，Eric酷爱音乐，其最新作品名为Immersion Station。这是他与环境音乐先锋Steve Roach合作的结晶，在iPhone应用商店有售。

Eric现与妻子和幼女居住在班布里奇岛。女儿经常前往其工作室，喜欢鼓捣那里的合音和音响器材的按钮。

联系Eric，可发送电子邮件至eric@wickedly-smart.com，也可访问其个人网站http://ericfree-man.com。

Elisabeth Robson 软件工程师、作者、培训师。获得耶鲁大学计算机科学硕士学位。在校期间就对技术产生了浓厚的兴趣，并设计了一款可视化并行编程语言和软件架构。

从职业生涯早期，她就致力于互联网领域的工作，与人一起打造了获奖网站The Ada Project。该网站首开先河，致力于帮助计算机领域的女性寻找工作和志同道合者。

当前，作为致力于传播Web技术的在线教育公司WickedlySmart的联合创始人，她负责图书、文章和视频等制作工作。Elisabeth曾担任O'Reilly Media的特殊项目主管，负责创办涉及各种技术主题的体验工作坊和在线课程，从此对技术培训产生了浓厚的兴趣。在此之前，Elisabeth就职于华特迪士尼公司，主要负责数字媒体的研发工作。

闲暇之余，Elisabeth要么投身于大自然，带着摄像机去徒步旅行、骑自行车或划皮划艇；要么在家烹制素食大餐。

联系Elisabeth，可发送电子邮件至beth@wickedlysmart.com，也可访问其博客http://elisabethrobson.com。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

目录（精简版）

前言	xxv
1 JavaScript速览：进入JavaScript的世界	1
2 编写代码：更进一步	43
3 函数简介：养成函数思维	79
4 让数据排排坐：数组	125
5 理解对象：对象镇之旅	173
6 与网页交互：了解DOM	229
7 类型、相等、转换等：系统地讨论类型	265
8 综合应用：编写一个应用程序	317
9 异步编码：处理事件	381
10 函数是一等公民：自由的函数	429
11 匿名函数、作用域和闭包：系统地讨论函数	475
12 高级对象构造技巧：创建对象	521
13 使用原型：超强的对象创建方式	563
附录 遗漏内容：未涉足的十大主题	623

目录（完整版）

前言

让大脑重视JavaScript。现在你正试着学习某些东西，为了不让学习卡壳，你的大脑也在帮忙，大脑在想：“最好把空间留给重要的事，比如什么动物是危险的？裸体滑雪是不是一个坏主意？”那么怎么才能欺骗你的大脑，让它认为学好JavaScript关系到你下半生的幸福呢？



本书适合你阅读吗	xxvi
我们知道你是怎么想的	xxvii
我们视读者为学习者	xxviii
元认知：对思维方式的思考	xxix
我们是如何做的	xxx
如何让大脑听话	xxxi
导读	xxxii
技术审校	xxxv
致谢	xxxvi

1 JavaScript速览

进入JavaScript的世界

JavaScript赋予你强大的力量。JavaScript是一款纯正的Web编程语言，让你能够给网页添加行为。有了JavaScript，你就能够与用户互动，响应有趣的事件，从网上收集数据并将其用于网页中，在网页中绘制图形等。网页不再是枯燥、乏味、静态的，不再只是一动不动地展现在你面前。掌握JavaScript后，你还能够赋予网页全新的行为。

JavaScript的工作原理 2
如何编写JavaScript 3
如何将JavaScript代码加入网页 4
JavaScript，你进步不小 6
如何编写语句 10
变量和值 11
切勿随意命名 12
自我表达 15
重复操作 17
while循环的工作原理 18
使用JavaScript进行决策 22
进行大量决策 23
与用户交流 25
console.log详述 27
打开控制台 28
编写一个正式的应用程序 29
如何将JavaScript代码加入网页（细数各种方式） 32
棒打鸳鸯散 33




WEB镇时报

如何避免令人难堪的命名错误

在给变量命名方面，你有极大的选择空间，下面是一些来自Web插的小提示，可帮助你更轻松地命名。

选择有意义的名称。

变量名 n、s、r或fcc对你来说也许是有意义的，但在Web精通常不受人待见。随着时间的推移，你可能忘记这些变量的含义，如果使用变量名angle、currentFrameNo和passwordExam，你的代码将更容易理解。

创建由多个单词组成的变量名时，采用驼峰式拼写法。

有时候，你可能需要一个变量，用于表示类似于喷火的双头龙（two-headed dragon with fire）这样的东西，如何给这样的变量命名呢？采用驼峰式拼写法即可，即将每个单词的首字母大写（第一个单词除外）：twoHeadedDragonWithFire。驼峰式拼写法被视为最简单、在Web插中被广泛使用，它提供了足够的可读性，创建的变量名要多具体就有多具体，还有其他命名方案，但这种方案更常用，在其他语言中亦是如此。

除非有充分的理由，否则不要使用以_或$打头的变量名。

以_打头的变量名通常保留用于JavaScript库，有些作者根据各种约定使用以_打头的变量名，然而，我们的建议是，除非有充分的理由（具体是什么理由你自己知道），否则不要使用这两种变量名。

小心驶得万年船。

给变量命名时，安全第一。本书后面还将介绍一些安全小贴士，就目前而言，为确保命名的安全，一定要避免使用关键字，声明变量时，一定要使用关键字var。

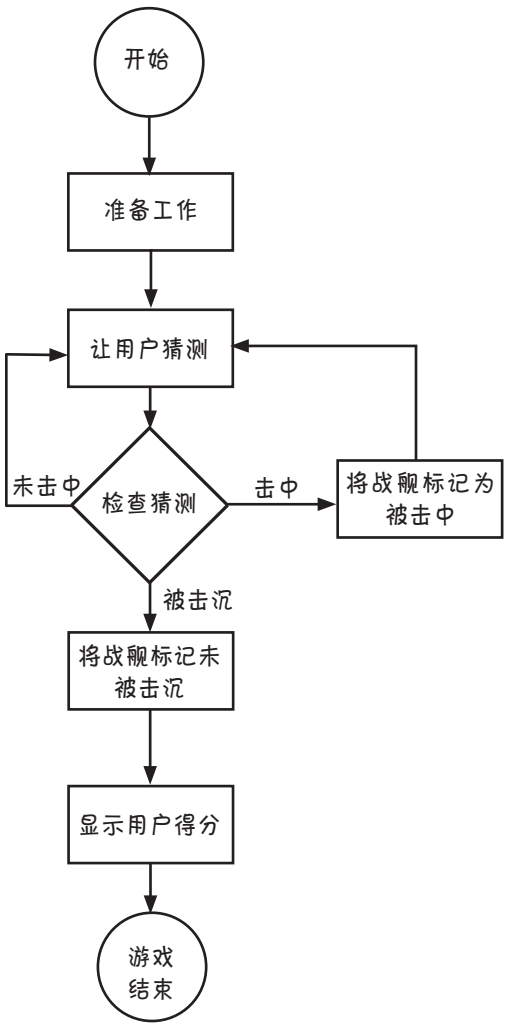


2 编写代码

更进一步

你已经知道了变量、类型、表达式等，我们接着往下介绍。

你对JavaScript有所了解，实际上已经具备了足够的知识，可以编写一些真正的代码了：一些完成有趣功能的代码，一些有人想使用的代码。你缺乏的是实际编写代码的经验，现在就来弥补吧。如何弥补呢？全身心地投入，开发一个完全使用JavaScript编写的休闲类游戏。我们目标远大，又脚踏实地，一步一个脚印。来吧，现在就开始。如果你想利用这款游戏来一次创业，我们不会有任何意见，因为代码是你的。



开发一款战舰游戏 44
第一次尝试——简化的战舰游戏 44
从高层次设计着手 45
编写伪代码 47
对了，别忘了HTML 49
编写简单的战舰代码 50
编写游戏逻辑 51
第一步：创建循环并获取输入 52
prompt的工作原理 53
检查用户的猜测 54
判断是否击中 56
添加击中检测代码 57
进行游戏后分析 58
完整的游戏逻辑 60
一点点质量保证 61
别这么啰嗦好不好 65
完善简单的战舰游戏 66
如何随机地指定位置 67
举世闻名的随机数生产配方 67
再来一点点质量保证 69
说说代码重用 71

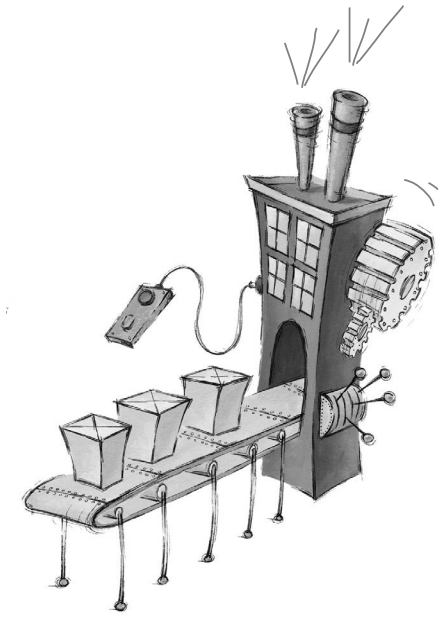
3 函数简介

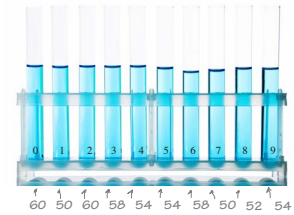
养成函数思维

为使用你的第一个超级武器作好准备。你编写过一些代码，现在该使用函数来提高效率了。通过使用函数，你可编写适用于各种不同环境的代码。这些代码可反复重用且管理起来容易得多。你还可以将通用代码抽取出来，给它指定一个简单的名称，这样就能将复杂的东西抛诸脑后，将精力放在重要的内容上。你将发现，函数不仅是脚本编写人员通往程序员的大门，还是JavaScript编程风格的核心。本章介绍函数的基本知识：机制以及工作原理的方方面面。在本书余下的篇幅中，你将不断提高函数方面的技能。下面就来为此打下坚实的基础吧。

这些代码到底有什么问题呢 81
以前提到过函数吗 83
函数到底是如何工作的呢 84
可以向函数传递哪些东西 89
JavaScript按值传递实参 92
错误地调用函数 94
函数还可返回值 95
详解包含return语句的函数的执行过程 96
全局变量和局部变量 99
了解局部变量和全局变量的作用域 101
短命的变量 102
千万别忘了声明局部变量 103







4 让数据排排坐

数组

在JavaScript中，并非只有数字、字符串和布尔值。前面编写JavaScript代码时，使用的都是基本类型（简单字符串、数字和布尔值），如"Fido"、23和true。使用基本类型可完成很多工作，但有时候必须处理更多数据，如购物车中的所有商品、播放列表中的所有歌曲、一系列恒星及其亮度或整个产品目录，为此，需要更强大的工具。对于这种按顺序排列的数据，可使用JavaScript数组来存储。本章将介绍如何将数据加入数组、如何传递数组以及如何操作数组。本书后面将介绍其他组织数据的方式，但现在先从数组着手吧。



你能帮助泡泡玩具反斗城公司吗	126
如何使用JavaScript表示多个值	127
数组的工作原理	128
确定数组的长度	130
自动造句应用程序	132
回到泡泡玩具反斗城公司	135
如何迭代数组	138
等等，还有一种更好的数组迭代方式	140
别这么啰嗦好不好	146
使用后递增运算符重新编写for循环	147
创建空数组并在其中添加元素	151
胜出的配方是……	155
快速审视代码	157
编写函数printAndGetHighScore	158
使用printAndGetHighScore重构代码	159
整合起来	161

5 理解对象

对象镇之旅

到目前为止，你在代码中使用的都是基本类型和数组。另外，你使用简单语句、条件、for/while循环和函数来编写代码，这种编码方式的过程化程度极高，不完全是面向对象的。实际上，这根本就不是面向对象的。你确实偶尔在不知不觉间使用了一些对象，但从未编写自己的对象。现在，该放弃这种枯燥的过程型编程方式，转而创建一些自己的对象了。在本章中，你将了解为何使用对象可让生活更美好，当然这是从编程意义上说的，因为本书只负责提高你的JavaScript技能，而不能让你变得更时尚。需要指出的是，到了对象镇，你会乐不思蜀，可别忘了给我们寄明信片。

对象 174
属性 175
如何创建对象 177
何为面向对象呢 180
属性的工作原理 181
变量是如何存储对象的 186
比较基本类型和对象 187
使用对象 188
分步进行预检 190
更深入地讨论向函数传递对象的情况 192
如何给对象添加行为 198
改进方法drive 199
为何方法drive不知道属性started 202
关键字this的工作原理 204
行为如何影响状态 210
让状态来影响行为 211
祝贺你熟悉了对象 213
对象无处不在（它们让你的生活更美好） 214

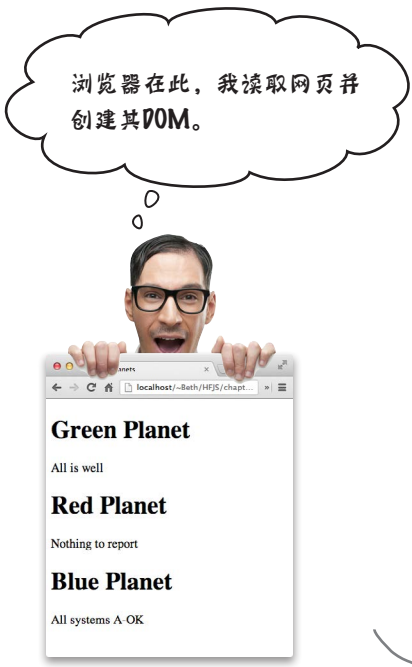


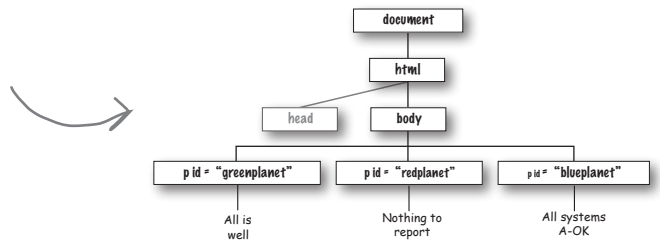
6 与网页交互

了解DOM

你的JavaScript水平有了很大的提高。事实上，你从门外汉成了脚本编写人员，又成了程序员，但还有一些东西没学。要充分利用你的JavaScript技能，就必须知道如何与代码所属的网页交互。只有这样，你才能编写出动态网页：能够对用户操作作出响应的网页，能够在加载后自动更新的网页。那么，如何与网页交互呢？使用DOM，即文档对象模型（document object model）。本章将详细介绍DOM，看看如何使用它和JavaScript赋予网页新功能。

上一章的密码破解难题 230
这些代码是做什么的呢 231
JavaScript如何与网页交互 233
如何自己动手制作DOM 234
初尝DOM 235
使用getElementById获取元素 240
从DOM获取的到底是什么 241
查找内部HTML 242
修改DOM的影响 244
围绕星球进行试驾 247
网页加载完毕前别想运行代码 249
你说“事件处理程序”，我说“回调函数” 250
如何使用setAttribute设置特性 255
获取特性 256
别忘了getElementById也可能返回null 256
DOM还有哪些功能 258

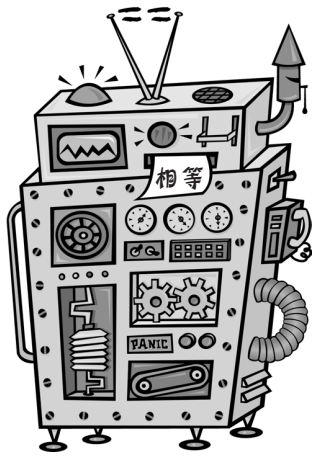




7 类型、相等、转换等

系统地讨论类型

该系统地讨论类型了。JavaScript的优点之一是，你无需知道这种语言的很多细节，就可以使用它来做很多事情。但要真正掌握这门语言、提升水平，做到人生中梦寐以求的事情，你必须对类型了如指掌。还记得本书前面是怎么说JavaScript的吗？是不是说它命不好，不是出生在高贵的学术殿堂，也未经同行审阅？确实如此，但非科班出身不妨碍史蒂夫·乔布斯和比尔·盖茨取得成功，对JavaScript来说亦如此。这确实意味着JavaScript的类型系统并不特别缜密，我们会发现它有一些怪癖。不过不用担心，本章将把这些怪癖解释得一清二楚，你很快就能避免各种与类型相关的难堪错误。



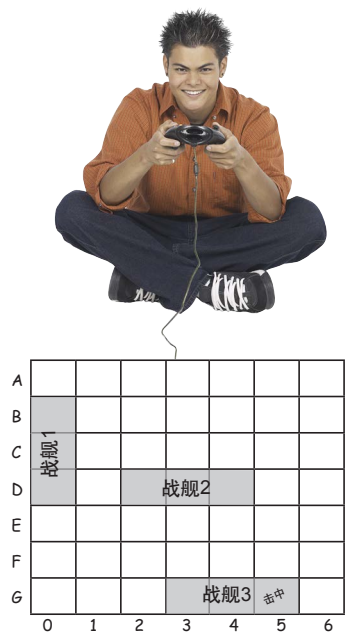
真相就摆在那里 266
小心，你可能意外遭遇undefined 268
如何使用null 271
处理NaN 273
比想象的还要怪异 273
有一点我们必须告诉你 275
理解相等运算符 276
相等运算符如何转换操作数（听起来比实际上危险） 277
严格相等 280
深入探讨类型转换 286
如何判断两个对象是否相等 289
真值就摆在那里 291
JavaScript将哪些值视为假值 292
字符串的神秘生活 294
为什么字符串既像基本类型又像对象 295
字符串方法和属性简明教程 297
座椅争夺战 301

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

8 综合应用

编写一个应用程序

系上工具腰带，就是那个囊括你新学的所有编码技能、DOM知识甚至HTML&CSS知识的工具腰带。本章将结合使用这些知识打造第一个货真价实的Web应用。不再是只有一艘战舰和一行藏身之地的小儿科游戏，本章将提供全面的体验：又大又漂亮的游戏板、多艘战舰、获取用户输入……这些都包含在一个网页中。我们将使用HTML定义游戏网页的结构，使用CSS设置游戏的视觉样式，并使用JavaScript代码定义游戏的行为。坐好了，我们将全力以赴，将油门踩到底，编写一些正式的代码。



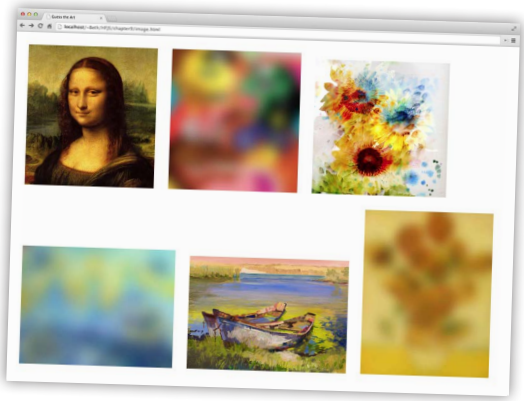
编写一个货真价实的战舰游戏 318
回顾HTML和CSS 319
创建HTML页面：大致轮廓 320
再添加一些样式 324
使用hit和miss类 327
如何设计这款游戏 329
实现View对象 331
displayMessage的工作原理 331
displayHit和displayMiss的工作原理 333
模型 336
如何表示战舰 338
实现model对象 341
实现方法fire 342
实现控制器 349
处理玩家的猜测 350
规划代码 351
实现parseGuess 352
记录猜测并开火的次数 355
如何给Fire!按钮添加事件处理程序 359
将输入交给控制器 360
如何放置战舰 364
编写方法generateShip 365
生成新战舰的起始位置 366
完成方法generateShip的编写工作 367

9 异步编码

处理事件

阅读完本章，你的编程方式再不似从前。到目前为止，你编写的代码通常都是按从上到下的顺序执行的。虽然你的代码要复杂些，还使用了一些函数、对象和方法，但从某种程度上说，它们都是按部就班地执行的。然而，JavaScript代码通常不是这样编写的，大多数JavaScript代码都是事件响应式的。很抱歉到现在才告诉你这一点。那么都是什么样的事件呢？用户单击网页、通过网络收到数据、定时器到期、DOM发生变化等。事实上，在浏览器中，幕后始终有各种事件在不断发生。本章将反思我们的编码方式，学习如何编写响应事件的代码以及为何要这样做。





事件是什么	383
事件处理程序是什么	384
如何创建第一个事件处理程序	385
测试事件处理程序	386
通过创建一个游戏来理解事件	388
实现游戏	389
试驾	390
再添加一些图像	394
将同一个事件处理程序赋给每幅图像的属性`onclick`	395
如何将同一个处理程序用于所有图像	396
事件对象的工作原理	399
使用事件对象	401
测试事件对象和`target`属性	402
事件和队列	404
其他事件	407
`setTimeout`的工作原理	408
完成猜图游戏	412
测试定时器	413

10 函数是一等公民



自由的函数

熟知函数你就能成为明星。无论在什么行业和学科，都存在让大师有别于芸芸众生的重要分水岭；就JavaScript而言，这个分水岭就是熟知函数。在JavaScript中，函数不可或缺，很多设计和组织代码的技巧都要求你精通并熟练地使用函数。通往精通函数的学习道路很有趣，但常常会让人迷惑不解，因此你一定要有心理准备。本章将更深入地介绍JavaScript函数，你将像刘姥姥进了大观园，见到一些奇异、古怪而神奇的东西。

关键字function神秘的双面人生 430
函数声明和函数表达式 431
分析函数声明 432
接下来呢？浏览器执行代码 433
继续往下执行 434
函数怎么也是值呢 439
我们提到过函数在JavaScript中是一等公民吗 442
乘坐头等舱 443
编写处理和检查乘客的代码 444
迭代所有的乘客 446
向函数传递函数 447
从函数返回函数 450
编写让乘客点饮料的代码 451
编写让乘客点饮料的代码：另一种方式 452
利用作为一等公民的函数让乘客点饮料 454
Web镇可乐公司 457
数组方法sort的工作原理 459
整合起来 460
测试排序代码 462



图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

11 匿名函数、作用域和闭包

系统地讨论函数

你已经全面了解了函数，但还需更深入地学习。本章将再进一步，深入函数的核心，演示如何娴熟地利用函数。这一章虽然不太长，但涵盖的知识点非常多；阅读完本章后，你的JavaScript表达能力将超乎你的想象。你还将为与人协作编写代码或使用开源JavaScript库作好准备，因为我们将介绍一些与函数相关的常见编码习惯和约定。如果你从未听说过匿名函数和闭包，那真是来对了地方。

函数的另一面 476
如何使用匿名函数 477
别这么啰嗦好不好 479
函数是在什么时候定义的？这要看情况 483
为何`fly`未定义 484
如何嵌套函数 485
嵌套对作用域的影响 486
词法作用域简介 488
词法作用域的有趣之处 489
再谈函数 491
再谈函数调用 492
闭包到底是什么 495
敲定函数 496
使用闭包实现神奇的计数器 498
揭秘 499
通过将函数表达式用作实参来创建闭包 501
闭包包含的是实际环境，而非环境的副本 502
使用事件处理程序来创建闭包 503
用作按钮单击处理程序的闭包的工作原理 506



12 高级对象构造技巧

创建对象

到目前为止，我们都以手动方式创建对象：对于每个对象，都使用对象字面量来指定其所有属性。小规模地创建对象时，这没有问题，但要编写正式的代码，需要使用更好的方式。这正是对象构造函数的用武之地。使用构造函数，可更轻松地创建对象，还可让所有对象都采用相同的设计蓝图；也就是说，使用构造函数可确保所有对象都包含相同的属性和方法。通过使用构造函数，编写的对象代码将简洁得多，而且创建大量对象时不容易出错。阅读完本章后，你谈论起构造函数，就会像是在对象镇长大的一样。

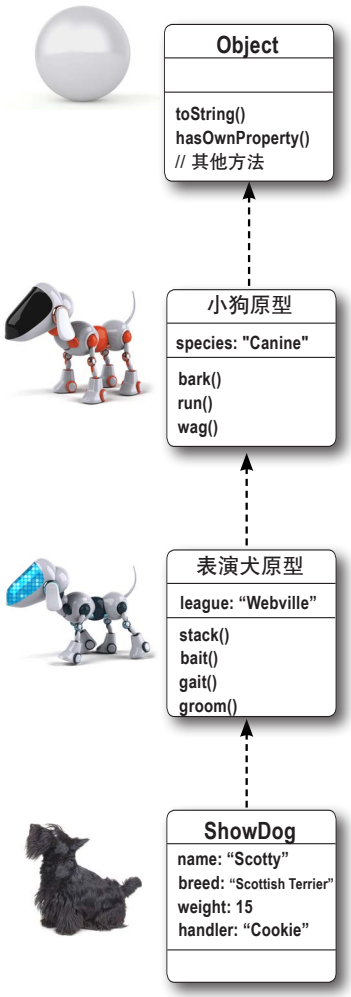


使用对象字面量创建对象 522
按约定创建对象 523
对象构造函数简介 525
如何创建构造函数 526
如何使用构造函数 527
构造函数的工作原理 528
还能在构造函数中定义方法 530
生产时间到了 536
试驾一些新车 538
不要将对象字面量弃若敝屣 539
用一个对象字面量替代所有实参 540
修改构造函数Car 541
理解对象实例 543
即便是创建好的对象，也可以有独特的属性 546
内置构造函数 548
数组对象 549
其他内置对象 551

13 使用原型

超强的对象创建方式

学会如何创建对象仅仅是个开始。该充实一些关于对象的内容了。我们需要在对象之间建立关系和共享代码的方法，需要扩展和改进既有对象。换句话说，我们需要更多的工具。在本章中你将看到，JavaScript的对象模型非常强大，但它与标准面向对象语言的对象模型稍有不同。JavaScript采用的不是基于类的面向对象系统，而是更强大的原型模型，其中的对象可继承和扩展其他对象的行为。这有何优点呢？你马上就会看到。现在就开始吧。



先来介绍一种更好的对象图	565
再谈构造函数：它能让我们重用代码，但效率如何呢	566
重复的方法真是个问题吗	568
原型是什么	569
继承原型	570
继承的工作原理	571
重写原型	573
如何设置原型	576
原型是动态的	582
方法sit更有趣的实现	584
再谈属性sitting的工作原理	585
如何设计表演犬	589
建立原型链	591
原型链中的继承原理	592
创建表演犬原型	594
创建表演犬实例	598
最后的整理	602
Dog.call详解	604
小狗原型并非原型链的终点	607
充分发挥继承的威力之重写内置行为	608
充分发挥继承的威力之扩展内置对象	610
JavaScript大统一理论	612
使用对象改善生活	612
整合起来	613
继续学习	613

A 附录　遗漏内容

未涉足的十大主题



我们介绍了大量的基本知识，本书也即将接近尾声。我们会想念你的。放手让你去独闯天涯前，我们想在你的行囊中再准备点东西，不然会不放心。这个附录篇幅较短，无法囊括你需要知道的一切。实际上，它最初确实涵盖了你需要知道但本书前面未介绍的一切JavaScript编程知识，但字体小得谁都看不清。因此，我们删除了其中的大部分内容，只留下最重要的十大主题。

等你阅读完这个附录后，本书就真的结束了。不过别忘了索引。（也是必读的！）

#1. jQuery 624

#2. 更多地使用DOM 626

#3. 对象`window` 627

#4. `arguments` 628

#5. 处理异常 629

#6. 使用`addEventListener`添加事件处理程序 630

#7. 正则表达式 632

#8. 递归 634

#9. JSON 636

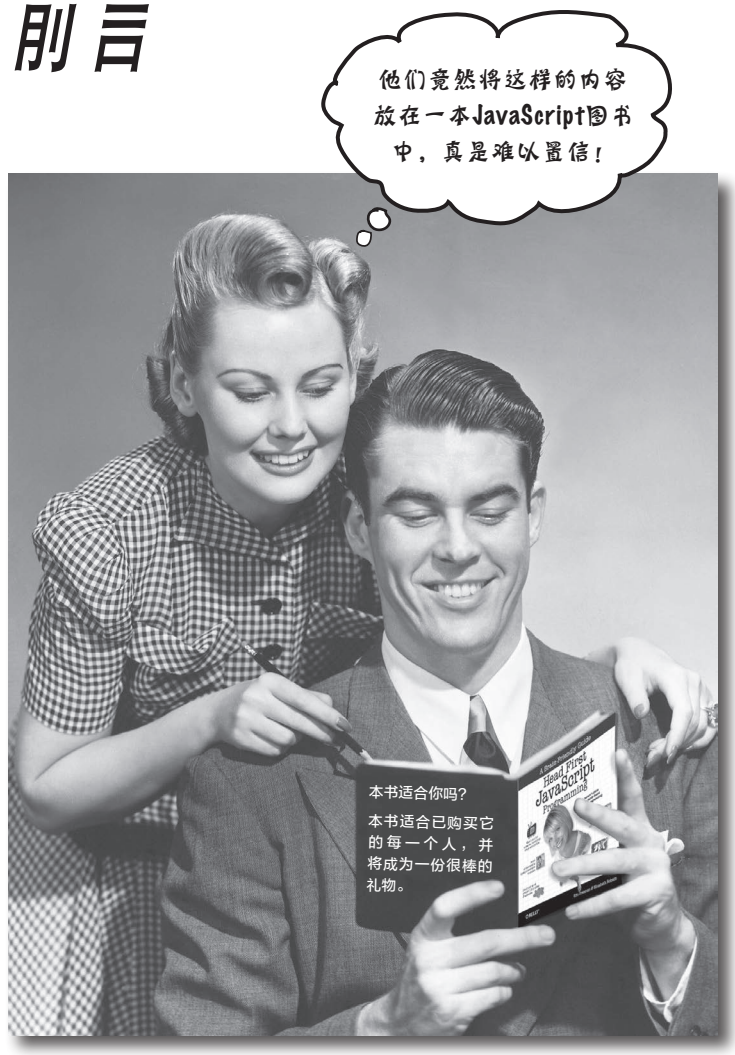
#10. 服务器端JavaScript 637

i 索引 639

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

如何使用本书

前言



这里将回答下面这个读者最关心的问题：在一本JavaScript图书中，为何包含这样的内容？

本书适合你阅读吗

如果对于下面的每个问题，你的回答都是肯定的，那么本书就是为你而写的。

1. 你有安装了**现代Web浏览器**和**文本编辑器**的计算机吗？

所谓现代Web浏览器，指的是最新版的Safari、Chrome和Firefox，以及9或更高版本的IE。

2. 你想**学习**、**理解**并**牢记**最佳的JavaScript**编程**技巧和最新的相关标准吗？

3. 相比于枯燥乏味的学术讲座，你更喜欢**晚宴上激动人心的交谈**吗？

市场部提示：只要有信用卡，任何人都可以购买本书。

你该对本书退避三舍吗

只要对下面的任何一个问题的回答是肯定的，你就应对本书退避三舍。

1. 你对Web开发一无所知吗？

   对你来说，HTML和CSS犹如天书吗？倘若如此，你也许应该先阅读《Head First HTML与CSS》，搞清楚如何编写网页，再考虑学习JavaScript。

2. 你是Web开发高手，想要的是一本**参考手册**吗？

3. 你**害怕尝试不同的东西**吗？你宁愿穿普通的衣服，也不穿格子配条纹的衣服吗？你认为既然JavaScript对象被赋予人性，那么技术书籍就应该一本正经吗？



我们知道你是怎么想的

“这书一点都不严肃。”

“那些图片都是干什么用的？”

“这样也能学会JavaScript吗？”

我们还知道大脑是怎么想的

大脑渴望新奇，它不断地寻找、扫描和等待不同寻常的东西。它天生如此，助你永葆活力。

这年头，你不太可能成为老虎的盘中餐，但你的大脑依然保持警惕，只是你从来没有意识到而已。

当你遇到寻常而普通的东西时，大脑是如何应对的呢？它竭尽所能，避免这些东西干扰它做重要的工作——记住重要的东西。它不会劳神去记住无聊的东西，因为这些东西显然是不重要的。

大脑怎么知道哪些东西重要呢？假设有一天你去徒步旅行，面前突然出现了一只老虎，你的大脑和身体会有何反应呢？

神经元受到刺激，情绪激动，肾上腺素激增。

大脑于是明白：

这一定很重要！可别忘了！

但假设在家里或图书馆学习、备考或学习老板认为需要一周甚至十天才能掌握的高深技术，此时你所处的环境很安全，根本不会有老虎。

大脑竭尽所能地帮助你，确保稀缺资源不会被显然不重要的内容占用，而将它们留给真正重要的东西，如老虎、火灾、不要穿着短袖短裤去滑雪等。

可是，你没法直截了当地向大脑发号施令：不管这本书有多乏味，也不管我当前对里氏震级有多不感兴趣，拜托你把这些内容记下来吧。

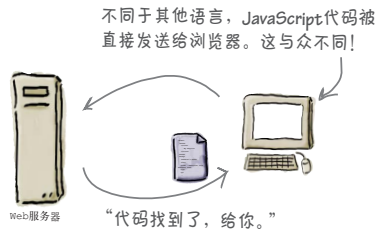


我们视读者为学习者

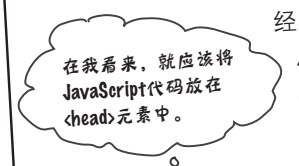
怎样才能学会东西呢？首先你得明白，其次你得牢记，但这并不意味着死记硬背。认知科学、神经生物学和教育心理学的最新研究表明，学习过程远比阅读文字要丰富得多。我们知道如何激活大脑。

本书遵循的一些学习原则

使用视觉化元素。图片比文字更容易记忆，并可极大地提高学习效率（可将记忆和理解程度提高89%）。图片还让知识理解起来更容易。通过**将文字放在相关图片的内部或附近，**而不是放在图片的下面或另一页，可将学习者解决相关问题的能力提高两倍。

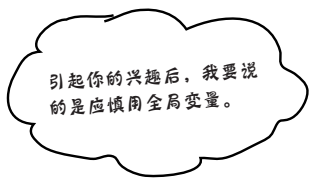


使用个性化的对话。最新的研究表明，相比于一本正经的语气，教师采用第一人称以对话的方式讲解时，学生的考试成绩可提高40%。我们使用口语化的语言讲故事，而不是一本正经地说教。激动人心的晚宴对话和枯燥讲座，哪个更能激发你的兴趣呢？



激发学习者更深入地思考。换而言之，如果你不去积极地刺激神经元，大脑就不会有太大的反应。要让学习者能够解决问题、得出结论、举一反三，就必须让他心生好奇、对学习有积极性并沉醉其中。为此，需要让学习者应对挑战、做练习、回答引人深思的问题，并完成一些需要左右脑和多个感官参与的活动。







引起并保持读者的兴趣。大家都有这样的经验：心里想学，可一页还没看完就瞌睡连连。大脑只会对不同寻常、有趣、怪异、抓人眼球、出乎意料的东西感兴趣。面对新的高深技术，如果学习过程不再枯燥乏味，就会事半功倍。

触动心弦。你知道，能不能记住一样东西，在很大程度上取决于它对你情绪的影响。你会记住自己关心的东西，也会记住触动你心弦的东西。这里说的可不是小男孩和小狗之间虐心的故事，而是在玩填字游戏、学习人人都认为很难的东西或意识到自己很厉害时油然而生的惊讶、好奇、开心、疑惑和傲视天下等情绪。



元认知：对思维方式的思考

如果你有心学习，还想学得更快、更深，就应专注于你如何才能做到专注，思考你的思维方式，了解你的学习方法。



大多数人在成长过程中都没有接触过元认知和学习理论。每个人都被要求学习，可知道如何学习的人少之又少。

我们认为，既然你手捧本书，肯定很想学习如何编写JavaScript程序。你可能不想花太多时间，而只想记住阅读的内容，并能够付诸应用。为此，你必须理解这些内容。为从本书（或任何图书或学习过程）获得最大的收获，就要对大脑负责，因为它将决定学习效果。

秘诀在于让大脑认为你学习的新内容非常重要，就像老虎一样生死攸关。不然，你的大脑将不断与你唱反调，不惜一切手段地排斥这些新内容。

如何让大脑认为JavaScript与老虎一样重要？

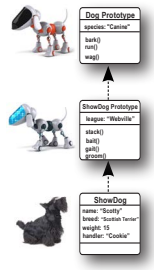
有缓慢乏味的方式，也有快速高效的方式。缓慢的方式就是没完没了地重复。你知道，即便是最无趣的主题，只要不断重复，就能让人学会并牢记在心。重复多了，你的大脑就会说：这个好像对他来说并不重要，可他看了又看，我只能认为这个肯定很重要了。

快速的方式是**竭尽所能地刺激大脑活动**，尤其是不同类型的大脑活动。“我们视读者为学习者”中提到的做法是刺激大脑活动的重要组成部分，实践表明，它们都有助于让大脑听话。例如，研究表明，通过将描述文字放在图片中，而不是图片说明或正文等其他地方，可让大脑明白文字和图片的关系。这将加大对神经元的刺激，而对神经元的刺激越大，就意味着大脑认为相关的内容值得关注，将其记住的可能性也就越大。

对话式风格可提高学习效率，因为人们意识到自己在参与对话时，就知道为避免对话中断，必须更加集中注意力。神奇的是，大脑并不介意你是在与图书对话。另一方面，如果写作风格一本正经而且乏味，大脑将以为你是坐在教室里听讲座，听众都抱着消极的态度，昏昏欲睡也没有关系。

然而，图片和对话风格只是冰山一角。

我们是如何做的



我们使用**图片**，因为大脑对视觉元素感兴趣，而对文字不感兴趣。在大脑看来，千言万语也抵不过一张图片。结合使用图片和文字时，我们将文字嵌在图片中，因为相比于出现在图片说明或正文中，描述文字出现在图片中可提高大脑的工作效率。

我们采用不同的方式**重复**同样的东西，通过不同的媒介刺激不同的感官，让这些内容留存在大脑的不同区域。

我们以你**意想不到**的方式使用概念和图片，因为大脑喜欢新奇。同时，使用图片和概念时，我们添加一些带有**感情色彩**的东西，因为大脑易受感情的支配。你更容易记住触动感情的东西，即便这种感情只是小小的**幽默**、**惊奇**或**趣味**。



变身浏览器

我们采用个性化的**对话风格**，因为相比于被动地倾听，当大脑认为你正在对话时，就会更加集中注意力，即便这种对话是在你和图书之间进行时亦是如此。

书中包含100多个**活动**，因为相比于纸上谈兵，当你真刀真枪地做事时，大脑更容易学会并记住知识。书中的练习既有挑战性，又在读者的能力范围内，因为这种练习是大多数人的最爱。



要点

我们采用了**多种学习风格**，因为有人喜欢循序渐进，有人喜欢先了解总体情况，还有人只想看代码示例。无论你是什么样的学习风格，都将受益于以多种不同方式展现的相同内容。

我们兼顾**左脑和右脑**，因为大脑越是沉醉其中，你学会并记住的可能性越大，同时能够集中注意力的时间越长。通过让左脑和右脑轮流工作和休息，可提高学习效率，延长学习时间。



填字游戏

我们从**不同的角度**展现**故事**和练习，因为在必须做出评估和判断时，大脑对知识的理解通常会更深。

书中包含一些比较有挑战性的**练习**和没有简单答案的**问题**，因为大脑活跃起来时，通常更容易学会并记住知识。这就像仅仅在健身房看人锻炼并不能塑身一样。我们竭尽所能，确保让你苦思冥想时，想的都是重要的问题，而不在难以理解的示例、充斥术语的难解文本上浪费脑力。



我们在故事、示例和图片等中使用**人物**，因为你也是人，你的大脑会更专注于人而不是物。

我们遵循80/20规则。我们假设你的目标是成为JavaScript开发高手，不会只读这一本书。有鉴于此，本书并非无所不包，只介绍了你真正需要的内容。



如何让大脑听话

我们该做的都做了，余下的就得靠你自己了。下面的小提示只是起点，你需要倾听大脑的心声，找出哪些做法对你来说管用，哪些不管用。去试试吧。

请将下面这部分剪下来，贴到冰箱上。

1 别着急。理解的越多，需要记忆的就越少。

不要只是埋头苦读，时不时地停下来想一想。看到问题时，不要马上去翻答案，要想象有人问你这个问题。大脑想得越深，学会并记住相关内容的可能性越大。

2 做练习，记笔记。

不能指望我们来替你做这些练习，不然，与让人替你锻炼何异。**拿起笔来，亲自动手。**大量的证据表明，在学习期间做些体力活动，可提高学习效率。

3 阅读“世上没有愚蠢的问题”。

一个都不要落下。**这些都是核心内容**，绝非可有可无的，千万不要跳过。

4 让阅读本书成为你睡前的最后一件事，至少不要做其他有挑战性的事情。

你放下书本后，学习过程（尤其是变成长期记忆的过程）还在继续。大脑需要消化的时间，如果在此期间你在做其他事情，就会忘记有些刚学到的知识。

5 多喝水。

身体水分充足时，大脑最清晰。如果身体缺少水分（可能你还未感到口渴时就会出现这种情况），人的认知能力就会下降。

6 大声读出来。

朗读会激活大脑的另一个部分。要理解什么东西或加深对其的记忆时，可将其大声朗读出来。更佳的做法是，尝试向人进行解释。这样你将学得更快，还可能发现阅读时没有发现的新东西。

7 倾听大脑的心声。

注意大脑是否不堪重负。如果你发现自己开始变得敷衍了事或看后就忘，就说明该休息了。疲劳到一定程度后，如果还不休息，就会欲速而不达，甚至影响学习兴趣。

8 心有感悟。

这样大脑才知道你阅读的内容很重要。置身到故事中，给图片添加说明。即便是抱怨笑话太蹩脚，也胜过毫无感触。

9 动手实践。

在新设计的项目中应用学到的知识，抑或改造既有的项目。只要动手实践就好，这样可获得一些做练习和活动时无法获得的经验。你需要的只是铅笔和需要解决的问题——一个可能受益于JavaScript的问题。

10 睡好觉。

为学习编程，需要在大脑中建立大量新的关联。多睡觉吧，这大有裨益。

导读

这是一本学习用书，而不是参考手册。本书所有的内容都是精心编写的，扫除了所有可能的拦路虎。但读第一遍时，你还是得从开头读起，因为本书假设你已经理解了前面介绍过的内容。

我们介绍JavaScript的优点，同时提醒你注意其不足之处。

JavaScript并非出身于象牙塔的语言，未经学术同仁长时间的审核。JavaScript应运而生，在早期的浏览器社区长大。因此需要注意，JavaScript有优点，也有不足。但总体而言，JavaScript相当出色，条件是你能妥善地使用它。

本书介绍如何充分发挥JavaScript的优势，同时会指出它的不足之处，让你能够规避。

我们并未介绍这门语言的每个方面。

对于JavaScript，需要学习的地方有很多。本书并非参考手册，而是学习指南，因此并未涵盖JavaScript的方方面面。我们的目标是教你JavaScript的基本知识，让你能够读懂任何参考手册，还能使用JavaScript做你想做的任何事情。

本书介绍的是在浏览器中执行的JavaScript。

浏览器不仅是最常见的JavaScript运行环境，还是最方便的运行环境（开始JavaScript编程的唯一要求是，一台安装了文本编辑器和浏览器的计算机）。在浏览器中运行JavaScript还意味着你能马上看到结果：编写代码后，只需重新加载网页，就能明白其功能。

本书提倡遵循最佳实践，确保代码结构良好、易于理解。

你希望编写的代码对所有的人来说都易于阅读和理解，并能在新的浏览器中正确运行。你希望以最为简单直接的方式编写代码，在完成工作的同时保质保量。本书将教你编写结构良好清晰并能适应未来变化的代码，编写让你自豪得都想裱起来挂在墙上的代码。

我们建议你阅读本书时使用多款浏览器。

我们教你编写符合标准的代码，但你依然可能发现，浏览器在解释JavaScript方面存在细微的差别。我们将竭尽所能，确保本书的代码在所有现代浏览器中都能正确地运行，并介绍一些确保代码得到所有现代浏览器支持的技巧。尽管如此，我们还是建议你至少使用两款浏览器对你编写的JavaScript代码进行测试。这让你能够了解不同浏览器的差别，并创建在各种浏览器中都能正确运行并得到相同结果的JavaScript代码。

编程是件严肃的事情，你得亲自动手，有时还得不辞辛劳。

如果你有一定的编程经验，就能明白这一点。你若刚读完《Head First HTML与CSS》，会发现编写代码有些不同。编程要求你采用不同的思维方式，它是一种逻辑性很强、有时甚至非常抽象的工作，要求你从算法的角度考虑问题。不过不用担心，我们的所有介绍都易于理解。只要一步一步来，并确保营养良好、睡眠充足，就能完全掌握这些新的编程概念。

活动都不是可有可无的。

在本书中，练习和活动并非附属品，而是核心内容的组成部分。有些活动和练习旨在帮助记忆，有些旨在帮助理解，还有一些旨在帮助你应用学到的知识。千万不要跳过。只有填字游戏并非是必做的，但它们能让你的大脑有机会从不同的角度思考这些单词。

重复是有意为之，而且很重要。

Head First系列丛书的独特之处在于，我们想让你完全理解学习的内容，读完后还能记得学到的知识。参考手册大多不以记忆为目的，但本书是一部学习指南，因此有些概念会反复介绍。

示例尽可能精简。

读者跟我们说，在一个200行的代码示例中寻找2行需要理解的代码真让人郁闷。本书的大多数示例都尽可能简化了背景，让要学习的代码简单明了。可别指望这些示例都完美无缺，因为这些代码是专为学习编写的，并非都功能完备。

为方便读者下载，我们将所有的示例文件都放到了网上：http://wickedlysmart.com/hfjs。

“考考你的脑力”练习并非都有解。

有些“考考你的脑力”练习没有正确的答案；有些是要让你判断自己的答案是否正确以及在什么情况下是正确的；还有些包含提示，旨在给你指明正确的方向。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

我们通常只列出JavaScript代码，而不列出HTML标记。

除前两章外，我们通常只列出JavaScript代码，并假定你将把它们嵌入到正确的HTML页面中。对于本书的大部分代码，你都可将其嵌入到下面这个简单的HTML页面中；要求将代码嵌入到其他HTML页面中时，我们会明确地告知。

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>Your HTML Page</title>
        <script>

                    ← JavaScript代码通常放在这里。

        </script>
    </head>
    <body>
            ← 网页内容都放在这里。

    </body>
</html>
```

不用担心，本书开头会带你完成所需的所有步骤。

获取代码示例、帮助和讨论。

阅读本书所需的一切都可在http://wickedlysmart.com/hfjs找到，其中包含代码示例文件以及视频等补充材料。

技术审校



Jeff Straw



Ismaël Martin "Bing" Demiddel

这些家伙一点都不含糊，他们参与了整个审阅过程，不放过每个角落，提供了宝贵而详尽的反馈。



Frank D. Moore



Alfred J. Speller



Bruce Forkush



Javier Ruedas

感谢出色的审阅团队

本书的审阅比我们以前编著的任何图书都详尽。事实上，有270多人加入了我们的WickedlySmart项目，实时地对本书进行审读和批评。这样做的效果超乎想象，对改善本书的各个方面大有裨益。这里衷心感谢每位审阅人员，是你们极大地提高了本书的品质。

上述技术审校都非常出色，他们还提供了其他的反馈，为本书作出了巨大的贡献。下面的审阅人员也为本书的各个方面作出了贡献：Galina N. Orlova、J. Patrick Kelley、Claus-Peter Kahl、Rob Cleary、Rebeca Dunn-Krahn、Olaf Schoenrich、Jim Cupec、Matthew M. Hanrahan、Russell Alleen-Willems、Christine J. Wilson、Louis-Philippe Breton、Timo Glaser、Charmaine Gray、Lee Beckham、Michael Murphy、Dave Young、Don Smallidge、Alan Rusyak、Eric R. Liscinsky、Brent Fazekas、Sue Starr、Eric (Orange Pants) Johnson、Jesse Palmer、Manabu Kawakami、Alan McIvor、Alex Kelley、Yvonne Bichsel Truhon、Austin Throop、Tim Williams、J. Albert Bowden II、Rod Shelton、Nancy DeHaven Hall、Sue McGee、Francisco Debs、Miriam Berkland、Christine H Grecco、Elhadji Barry、Athanasios Valsamakis、Peter Casey、Dustin Wollam和Robb Kerley。

致谢*

非常感谢受人尊敬的技术审校DavidPowers。事实上，我们编著图书再也离不开David了，他救了我们无数次。David之于我们，犹如擅长作词的Bernie之于作曲家Elton；这让我们禁不住扪心自问，没有他我们还能编著图书吗？David帮助敦促我们将本书写得更出色、技术上更准确，他的第二职业是脱口秀演员，在我们改进本书较为幽默的部分时帮了大忙。再次感谢David——你如此专业，知道通过了你的技术审核后，我们连睡觉都更安稳了。

受人尊敬的技术审校
David Powers



可别被他的笑容蒙骗了，这可是个吹毛求疵的家伙（当然是就技术上而言）。



Meghan Blanchette

O'Reilly的编辑

还要万分感谢编辑Meghan Blanchette为本书的出版铺平道路并扫除所有的障碍，感谢她耐心地等待并牺牲与家人团聚的时光。她还让我们在和O'Reilly处理彼此关系时保持理智。我们太喜欢你了，迫不及待再次与你合作！

特别感谢荣誉主编Mike Hendrickson提议编写本书。再次感谢Mike，如果没有你，我们一本书都出版不了。十多年来，你一直是我们的头号功臣，真是太爱你了！



Mike Hendrickson

*之所以在致谢部分提到这么多人，是因为我们想验证一个理论：书中致谢提到的每一个人都至少会买一本书，他们的亲戚朋友也可能会购买。如果你想让自己的名字出现在我们下本书的致谢部分，而且有一个大家族，请联系我们。

O’Reilly团队

还要衷心感谢O’Reilly团队。感谢Melanie Yarbrough、Bob Pfahler和Dan Fauxsmith群策群力，让本书得以付梓；感谢Ed Stephenson、Huguette Barriere和Leslie Crandell在营销工作上身先士卒，我们很喜欢他们的创造性营销方式；感谢Elli Volkhausen、Randy Comer和Karen Montgomery充满灵感的封面设计，一如既往地让我们非常满意；感谢Rachel Monaghan所做的文字加工工作，既一丝不苟，又保留了幽默的风格；感谢Bert Bates提供宝贵的反馈。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

1 JavaScript速览

进入JavaScript的世界



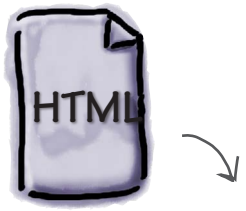
JavaScript赋予你强大的力量。JavaScript是一款纯正的Web**编程语言**，让你能够给网页**添加行为**。有了JavaScript，你就能够与用户互动，响应有趣的事件，从网上收集数据并将其用于网页中，在网页中绘制图形等。网页不再是枯燥、乏味、静态的——只是一动不动地躺在那里。掌握JavaScript后，你还能够赋予网页**全新**的行为。

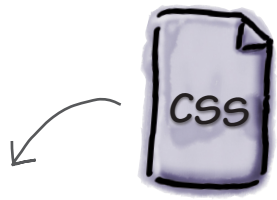
你也不会只是一个人在战斗。JavaScript是**最流行**的编程语言之一，所有现代浏览器（和大多数古老的浏览器）都**支持**它；其外沿在不断延伸，正逐渐被**嵌入**除浏览器外的其他众多环境中。话不多讲，咱们现在就动手吧！

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

JavaScript的工作原理

如果你习惯于在网页中添加结构、内容、布局和样式，现在也该添加一些行为了！这年头，只是静静地躺在那里的网页已经没有市场。要给人留下深刻的印象，网页必须是动态、交互性的，并以新颖的方式与用户互动；这正是JavaScript的用武之地。下面先来看看JavaScript在**网页生态系统**中所处的位置。

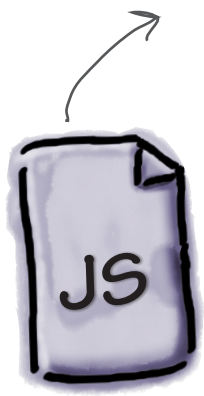




你已经知道，我们使用HTML（Hypertext Markup Language，超文本标记语言）来指定网页的**内容**和**结构**（如段落、标题和区块）。



你还知道，我们使用CSS（Cascading Style Sheets，层叠样式表）来指定网页的外观：颜色、字体、边框、边距和网页的布局。CSS用于指定**样式**，这是以独立于网页结构的方式实现的。



JavaScript是HTML和CSS负责计算的兄弟，下面简要地介绍一下它。JavaScript让你能够在网页中添加**行为**。要在用户单击On Sale for the next 30 seconds!按钮时做出响应吗？要即时地检查用户的表单输入吗？要从Twitter提取并显示消息吗？要在网页中运行游戏吗？求助于JavaScript吧。JavaScript让你能够在网页中进行编程，从而实现计算、响应、绘画、通信、提醒、变更、更新、修改等动态功能。这些都是JavaScript的用武之地。

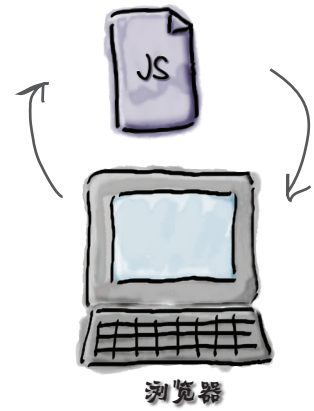
图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

如何编写JavaScript

JavaScript在编程领域独树一帜。使用典型的编程语言时，你必须先编写代码，再编译、链接和部署。JavaScript则灵活多变得多。使用JavaScript时，你只需直接在网页中编写代码，再在浏览器中加载网页，浏览器就会愉快地执行你编写的代码。下面更深入地探索其中的工作原理。







编写 ❶

你像往常那样创建网页：使用HTML指定内容，使用CSS指定样式，同时在网页中添加JavaScript代码。稍后你将看到，与CSS一样，可将JavaScript放在网页中，也可将其放在独立的文件中，并在网页中包含该文件。

加载 ❷

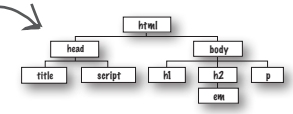
与往常一样，在浏览器中输入网页的地址。浏览器遇到JavaScript代码后，将立即对其进行分析，为执行做好准备。与HTML和CSS一样，发现JavaScript代码存在错误时，浏览器会尽力继续读取后面的JavaScript、HTML和CSS，尽可能避免无法向用户显示网页的情况发生。

执行 ❸

在网页中遇到JavaScript代码后，浏览器立即执行它们，并在网页的整个生命周期内不断地执行。不同于早期的JavaScript版本，当今的JavaScript使用了高级编译技术，其动力强劲，执行代码的速度几乎能够与原生编程语言媲美。

稍后将讨论哪种做法最合适。

浏览器还会创建HTML页面的对象模型，可供JavaScript使用。这将在后面更详细地介绍，现在请将其暂时抛在脑后。



如何将JavaScript代码加入网页

重要的事情先说。如果不知道如何将JavaScript代码加入网页，你在JavaScript的道路上就走不了多远。如何加入呢？当然是使用<script>元素！

下面来看一个枯燥而平凡的网页，并使用<script>元素添加一些动态行为。请不要过度关注我们将添加的<script>元素的细节，你当前的目标是运行一些JavaScript代码。

这是标准的HTML5 doctype以及<html>和<head>元素。

这个页面的<body>元素也相当普通。

在这个网页的<head>元素中，我们添加了一个<script>元素。

在<script>元素中，我们编写了一些JavaScript代码。

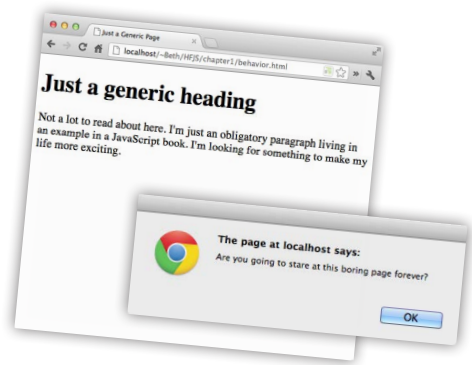
再说一遍，不要过度关注这些代码的功能。但我敢打赌，你肯定会浏览这些代码，并猜测各个部分的功能。

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Just a Generic Page</title>
    <script>
      setTimeout(wakeUpUser, 5000);
      function wakeUpUser() {
         alert("Are you going to stare at this boring page forever?");
      }
    </script>
  </head>
  <body>
      <h1>Just a generic heading</h1>
      <p>Not a lot to read about here. I'm just an obligatory paragraph living in
an example in a JavaScript book. I'm looking for something to make my life more
exciting.</p>
  </body>
</html>
```

试驾



将这个网页的内容输入到一个名为behavior.html的文件中，将其拖放到浏览器中（或选择菜单“文件”>“打开”），以加载它。其中的JavaScript代码有何功能呢？你需要等待5秒钟才能知道。





一定要放松心情。我们并不要求你现在就对JavaScript了如指掌。事实上，我们只想让你感受一下JavaScript是什么样的。

话虽如此，你也不能完全置身事外，因为我们想让你热热身，让大脑活跃起来。还想着前一页的代码？下面就来猜猜它们是做什么的：

创建可重用的代码并将这些代码命名为wakeUpUser？

```
setTimeout(wakeUpUser, 5000);
function wakeUpUser() {
   alert("Are you going to stare at this boring page forever?");
}
```

可能是一种数5秒的方式？提示：1000毫秒 = 1秒。

显然是一种使用消息提醒用户的方式。

世上没有愚蠢的问题

问：我听说JavaScript是一款无用的语言，是这样的吗？

答：刚面世时，JavaScript的动力肯定谈不上有多强劲，但随后其对Web来说变得日益重要，因此很多人（包括行业中一些最优秀的人才）都为改善JavaScript的性能做出了努力。不过你可能不知道，即便是在速度变得超快前，JavaScript也是一款非常杰出的语言。你将看到，我们会使用它来实现一些非常强大的功能。

问：JavaScript与Java有关系吗？

答：除名字外毫无关系。JavaScript推出时，Java已经是炙手可热的流行语言，为搭上Java这辆顺风车，JavaScript的发明者在其名称中包含了Java。这两种语言都借鉴了C等编程语言的一些语法，但除此之外，它们有天壤之别。

问：使用JavaScript是创建动态网页的最佳方式吗？诸如Flash等解决方案怎么样？

答：Flash一度是很多人开发交互式网页和动态网页的首选，但行业的发展天平正日益倾向于HTML5和JavaScript。HTML5推出后，JavaScript已成为标准的Web脚本语言。很多人都在努力改善JavaScript的性能和效率，以及开发扩展浏览器功能的JavaScript API。

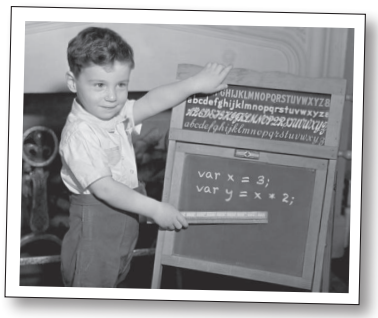
问：我的朋友在Photoshop中使用JavaScript，至少他是这样说的。这可能吗？

答：完全可能。作为一种通用的脚本语言，JavaScript的使用范围不再局限于浏览器，还用于从图形工具到音乐应用程序的众多应用程序中，甚至用于服务器端编程。通过学习JavaScript，你的付出在未来很可能在除网页外的其他领域得到回报。

问：你说很多其他语言都是编译型的。这到底是什么意思呢？为何JavaScript不是编译型的？

答：使用C、C++或Java等传统编程语言时，执行代码前必须进行编译。编译是将代码转换为适合计算机的表示方式，通常可改善运行阶段性能。脚本语言通常是解释型的，这意味着浏览器执行它遇到的每行JavaScript代码。脚本语言不那么看重运行阶段性能，而更强调灵活性，因此更适合用于完成原型开发和交互式编码等任务。JavaScript最初是解释型的，因此多年来其性能始终不那么高。然而，还有一条中间路线，即可对解释型语言进行即时编译，这正是浏览器厂商对现代JavaScript采取的做法。事实上，现在使用JavaScript既可获得脚本语言的便利性，又可享受编译型语言的性能。顺便说一句，本书将使用术语解释、评估和执行，在不同的上下文中，它们的含义存在细微的差别，但在本书中，它们基本上是一回事。

JavaScript，你进步不小



JavaScript 1.0

Netscape是第一家真正意义上的浏览器公司，它面世时你可能还没有出生。20世纪90年代中期，浏览器领域的竞争异常惨烈，与微软的竞争尤其如此，因此给浏览器添加激动人心的新功能成了重中之重。

为此，Netscape想推出一款脚本语言，让任何人都能够在网页中添加脚本。LiveScript应运而生，这是一款为迅速满足上述需求而开发的语言。你可能从未听说过LiveScript，这是因为当时Sun公司推出了Java，其股票的价格一飞冲天。为搭上Java成功的顺风车，Netscape将LiveScript改名为JavaScript。虽然JavaScript和Java一点关系都没有，可谁在乎呢？

微软有什么反应呢？它紧跟着Netscape推出了自己的脚本语言JScript，这种语言与JavaScript很像。浏览器之战就此爆发。

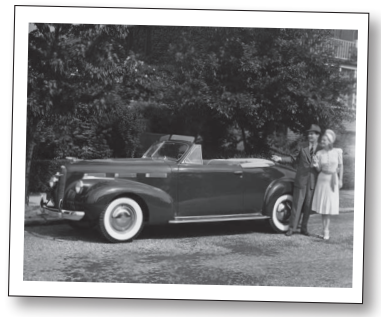


JavaScript 1.3

从1996年到2000年，JavaScript逐渐成熟起来了。实际上，Netscape提交了对JavaScript进行标准化的申请，ECMAScript由此诞生。从未听说过ECMAScript？没关系，你现在听说过了；你只需知道它是所有JavaScript实现（无论是浏览器还是其他环境中）的标准语言定义。

在此期间，受浏览器之战的影响，开发人员使用JavaScript时依然举步维艰，因为不同浏览器之间存在众多的差别，但几乎在所有情况下，JavaScript都成了不二的选择。与此同时，JavaScript和JScript之间虽然存在细微的差别，让开发人员头痛不已，但随着时间的推移，这两种语言变得越来越像。

彼时JavaScript还未摆脱业余语言的名声，但情况很快就会发生变化。



JavaScript 1.8.5

JavaScript终于成熟起来，获得了专业开发人员们的尊敬！你可能会说，这都是拜强大的标准（比如当前所有现代浏览器都遵循的ECMAScript 5）所赐，但实际上JavaScript得以用于专业领域，都是Google推动的结果。2005年，Google发布了Google Maps，向全世界展示了JavaScript在创建动态网页方面的强大威力。

随着JavaScript再次成为关注的焦点，很多杰出的编程语言人员致力于改善JavaScript解释器，极大地改善了其运行阶段性能。与面世之初相比，现今的JavaScript变化不大；它急匆匆地来到这个世界，却是一款功能和表现力强大的语言。



图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权



你看看，编写JavaScript多容易

你虽然还不熟悉JavaScript，但肯定能够对一些JavaScript代码的作用猜个八九不离十。请看下面的每行代码，你能猜出它们的作用吗？请在下面写出你的答案。我们指出了第一行代码的功能。如果你猜不出来，请参阅下一页的答案。

```
var price = 28.99;
var discount = 10;
var total =
      price - (price * (discount / 100));
if (total > 25) {
   freeShipping();
}

var count = 10;
while (count > 0) {
   juggle();
   count = count - 1;
}

var dog = {name: "Rover", weight: 35};
if (dog.weight > 30) {
    alert("WOOF WOOF");
} else {
    alert("woof woof");
}

var circleRadius = 20;
var circleArea =
   Math.PI * (circleRadius * circleRadius);
```

创建一个名为price的变量，并将28.99赋给它



你看看，编写JavaScript多容易

你虽然还不熟悉JavaScript，但肯定能够对一些JavaScript代码的作用猜个八九不离十。请看下面的每行代码，你能猜出它们的作用吗？请在下面写出你的答案。我们指出了第一行代码的功能。下面就是给出的答案。

`var price = 28.99;`	创建一个名为price的变量，并将28.99赋给它
`var discount = 10;`	创建一个名为discount的变量，并将10赋给它
`var total =` `  price - (price * (discount / 100));`	根据折扣计算新的价格，并将其赋给变量total
`if (total > 25) {`	将变量total的值与25进行比较。如果它大于25，
`  freeShipping();`	就使用freeShipping执行相应的操作
`}`	if语句到此结束
`var count = 10;`	创建一个名为count的变量，并将10赋给它
`while (count > 0) {`	只要变量count大于0，
`  juggle();`	就变变戏法，
`  count = count - 1;`	并将count的值减1
`}`	while循环到此结束
`var dog = {name: "Rover", weight: 35};`	创建一个包含名字和体重的dog对象
`if (dog.weight > 30) {`	如果该dog对象的体重大于30，
`  alert("WOOF WOOF");`	就在网页中显示WOOF WOOF
`} else {`	否则，
`  alert("woof woof");`	就在网页中显示woof woof
`}`	if/else语句到此结束
`var circleRadius = 20;`	创建一个名为circleRadius的变量，并将20赋给它
`var circleArea =`	创建一个名为circleArea的变量，
`  Math.PI * (circleRadius * circleRadius);`	并将这个表达式的结果（1256.6370614359173）赋给它



确实如此。

使用HTML和CSS，可创建一些漂亮的网页，但熟悉JavaScript后，就可以创建其他类型的网页了。实际上，你甚至可以将网页视为应用程序（甚至是一种体验），而不仅仅是网页。

这通常还能获得更高的薪水！

你可能会说：“我知道，确实如此。不然我为什么会阅读这本书呢？”实际上，我们是想借此机会谈谈如何学习JavaScript。如果你使用过编程语言或脚本语言，肯定大致知道即将面临的困难。如果你以前主要使用的是HTML和CSS，就必须明白编程语言的一些截然不同之处。

使用HTML和CSS时，你做的主要工作都是声明型的，例如指出一些文本为段落，或指出属于`sale`类的元素都为红色。使用JavaScript时，则要给网页添加行为，为此需要对计算进行描述。你需要知道如何描述类似于下面的事情：对所有的正确答案求和，以计算用户的得分；用户单击按钮时，播放表示获胜的声音；取回我最近发布的消息，并将其放到这个网页中。

为此，需要使用与HTML和CSS截然不同的语言。下面就来看看如何使用这种语言。

如何编写语句

创建HTML时，你通常对文本进行**标记**，以指定其结构。为此，要给文本添加元素、特性和值。

```
<h1 class="drink">Mocha Caffe Latte</h1>

<p>Espresso, steamed milk and chocolate syrup,
just the way you like it.</p>
```

我们使用HTML对文本进行标记，以指定其结构。例如，我需要一个名为Mocha Caffe Latte的大标题，它是一种饮料的标题。接下来，我需要一个段落。

CSS稍微有点不同。使用CSS时，你编写一系列**规则**，其中每条规则都指定了网页中的元素及其样式。

使用CSS时，我们编写规则，这些规则使用选择器（如h1.drink和P）指定要将样式应用于HTML的哪些部分。

```
h1.drink {
    color: brown;
}
p {
    font-family: sans-serif;
}
```

将所有饮料的标题都显示为棕色。

所有段落都使用sans-serif字体。

使用JavaScript时，你编写**语句**。每条语句都指定了计算的一小部分，而所有语句一起给网页添加行为。

一系列语句。

每条语句都做一点点工作，如声明用于存储值的变量。

```
var age = 25;
var name = "Owen";

if (age > 14) {
    alert("Sorry this page is for kids only!");
} else {
    alert("Welcome " + name + "!");
}
```

这里创建了两个变量，它们分别包含年龄25和值Owen。

有些语句作决策，如用户的年龄是否大于14。

如果是，就提醒用户，指出他太大了，该网页对他来说不合适。

否则，就欢迎用户，如显示Welcome Owen!。（但在这里Owen的年龄为25，因此不会这样显示。）

变量和值

你可能注意到了，JavaScript语句通常包含变量。变量用于存储值。什么样的值呢？下面是一些示例。

```
var winners = 2;
```

这条语句声明一个名为winners的变量，并将数字值2赋给它。

```
var name = "Duke";
```

这条语句将一串字符（简称为字符串）赋给变量name。

```
var isEligible = false;
```

这条语句将值false赋给变量isEligible。ture/false被称为布尔值。

请注意，布尔值无需用引号括起。



除数字、字符串和布尔值外，变量还可存储其他类型的值，这将在稍后介绍；但不管存储的是哪种类型的值，变量的创建方式都相同。下面更深入地介绍如何声明变量。

声明变量时，总是以关键字var打头。

任何情况下都应这样做，尽管省略关键字var，JavaScript也不会报错。后面将解释其中的原因。

接下来，给变量指定名称。

```
var winners = 2;
```

赋值语句总是以分号结束。

在声明变量时，也可给它赋值，方法是在变量名后面加上等号和值。

这里说“也可”是因为创建变量时，可以不给它指定初始值，而在以后再给它赋值。要在创建变量时不指定初始值，只需省略赋值部分即可，如下所示。

省略等号和值时，只是声明了一个可供以后使用的变量。

```
var losers;
```



切勿随意命名

你知道变量有名字，你也知道变量有值。

你还知道，变量可存储数字、字符串和布尔值等。

但如何给变量命名呢？随便命名都行吗？不是这样的，但变量命名规则非常简单：只要遵循下面两条规则，指定的变量名就是合法的。

1. 以字母、下划线或美元符号打头。
2. 然后使用任意数量的字母、数字、下划线或美元符号。

哦，还有一点：不能使用任何内置关键字，如`var`、`function`、`false`等，以免让JavaScript感到迷惑。因此，给变量命名时，务必远离这些禁区。下面列出了JavaScript关键字，本书后面将使用其中的一些，并指出它们的含义：

break	delete	for	let	super	void
case	do	function	new	switch	while
catch	else	if	package	this	with
class	enum	implements	private	throw	yield
const	export	import	protected	true	
continue	extends	in	public	try	
debugger	false	instanceof	return	typeof	
default	finally	interface	static	var	



世上没有愚蠢的问题

问：何为关键字？

答：关键字是JavaScript保留字，JavaScript将其用于特殊目的。如果将关键字用作变量名，将让你和浏览器都感到迷惑。

问：如果将关键字用作变量名的一部分呢？例如，可将变量命名为`ifOnly`（即在变量名中包含关键字`if`）吗？

答：当然可以，只要变量名不与关键字完全相同就行。另外，编写代码时应确保其清晰，因此通常不要使用变量名`elze`，因为它容易与`else`混淆。

问：JavaScript区分大小写吗？换句话说，`myvariable`和`MyVariable`指的是一回事吗？

答：如果你习惯了HTML标记，可能习惯于不区分大小写，因为在浏览器看来，<head>和<HEAD>是一回事。然而，在JavaScript中，变量名、关键字和函数名等几乎所有一切的大小写都很重要，因此请务必注意大小写。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权



WEB镇时报

如何避免令人难堪的命名错误

在给变量命名方面，你有极大的选择空间，下面是一些来自Web镇的小提示，可帮助你更轻松地命名。

选择有意义的名称。

变量名_ m、$、r或foo对你来说也许是有意义的，但在Web镇中通常不受人待见。随着时间的推移，你可能忘记这些变量的含义；如果使用变量名angle、currentPressure和passedExam，你的代码将更容易理解。

创建由多个单词组成的变量名时，采用骆驼式拼写法。

有时候，你可能需要一个变量，用于表示类似于喷火的双头龙（two-headed dragon with fire）这样的东西。如何给这样的变量命名呢？采用骆驼式拼写法即可，即将每个单词的首字母大写（第一个单词除外）：twoHeadedDragonWithFire。骆驼式拼写法使用起来很简单，在Web镇中被广泛使用，它提供了足够的灵活性，创建的变量名要多具体就有多具体。还有其他命名方案，但这种方案更常用，在其他语言中亦是如此。

除非有充分的理由，否则不要使用以_或$打头的变量名。

以$打头的变量名通常保留用于JavaScript库；有些作者根据各种约定使用以_打头的变量名。然而，我们的建议是，除非有充分的理由（具体是什么理由你自己知道），否则不要使用这两种变量名。

小心驶得万年船。

给变量命名时，安全第一。本书后面还将介绍一些安全小贴士。就目前而言，为确保命名的安全，一定要避免使用关键字；声明变量时，一定要使用关键字var。



有趣的语法

- 每条语句都以分号结尾。
```
x = x + 1;
```
- 单行注释以两个斜杆打头。注释只是用于向你或其他开发人员解释代码，它们不会执行。
```
// I'm a comment
```
- 空白无关紧要（几乎在什么地方都是如此）。
```
x      =      2233;
```
- 将字符串用双引号括起（也可使用单引号，这两种方式都可行，只要统一即可）。
```
"You rule!"
'And so do you!'
```
- 不要用引号括起布尔值true和false。
```
rockin = true;
```
- 声明变量时，可以不给它指定值。
```
var width;
```
- 不同于HTML标记，JavaScript区分大小写，这意味着大小写很重要。counter和Counter是两个不同的变量。

变身浏览器

下面的JavaScript代码存在一些错误。你的任务是变身浏览器，将其中的错误找出来。完成这个练习后，请翻到本章末尾，看看你是否找出了所有的错误。



A

```
// 笑话测试
var joke = "JavaScript walked into a bar....';
var toldJoke = "false";
var $punchline =
  "Better watch out for those semi-colons."
var %entage = 20;
var result

if (toldJoke == true) {
    Alert($punchline);
} else
    alert(joke);
}
```

请不要过度关注这些JavaScript代码的作用，重点是变量名错误和语法错误。

B

```
\\ 电影之夜
var zip code = 98104;
var joe'sFavoriteMovie = Forbidden Planet;
var movieTicket$    =    9;

if (movieTicket$ >= 9) {
    alert("Too much!");
} else {
    alert("We're going to see " + joe'sFavoriteMovie);
}
```

自我表达



要使用JavaScript准确地表达自我，需要使用**表达式**。表达式的结果为值，前面的代码示例就包含多个表达式，下面是其中之一：

这是一条JavaScript语句，它将一个表达式的结果赋给变量total。

*表示乘法运算，而/表示除法运算。

```
var total = price - (price * (discount / 100));
```

这是变量total。

这是赋值运算符。

这是一个表达式。

这个表达式计算打折后的价格（price），其中的折扣（discount）是一个百分比值。因此，如果价格为10，折扣为20，则结果为8。

如果你学过数学、计算过收支或计算过税费，就不会对这样的数值表达式感到陌生。

还有字符串表达式，如下所示：

```
"Dear " + "Reader" + ","
```

将这些字符串相加（拼接），得到新字符串“Dear Reader,”。

```
"super" + "cali" + youKnowTheRest
```

与前一个表达式类似，但包含一个字符串变量。这个表达式的结果为“supercalifragilisticexpialidocious”。*

```
phoneNumber.substring(0,3)
```

另一个结果为字符串的表达式。它返回表示美国电话号码的字符串中的区号，其中的工作原理将在后面介绍。

还有结果为true或false的表达式，这种表达式称为布尔表达式。请看下面的各个表达式，它们返回true还是false呢？

```
age < 14
```

如果age小于14，这个表达式就为true，否则为false。可以使用这个表达式来判断一个人是不是孩子。

```
cost >= 3.99
```

如果cost等于或大于3.99，这个表达式就为true，否则为false。如果这个表达式为false，就准备好购买减价商品吧！

```
animal == "bear"
```

变量animal包含字符串bear时，这个表达式为true。如果是这样，可得小心了！

表达式的值还可能为其他几种类型，将在本书后面介绍。就现在而言，重点是牢记所有表达式的结果都为某种值：数字、字符串或布尔值。下面来看看表达式都有哪些作用。

* 假定变量youKnowTheRest的值为“fragilisticexpialidocious”。



磨笔上阵

请拿起笔来，将一些表达式的作用搞清楚。计算下面每个表达式的值并写下来。是的，写下来。将老妈对你不要在书上乱写乱画的嘱咐忘了吧，在本书中想怎么写就怎么写！一定对照本章末尾的答案，看看你的答案对不对。

这是一个摄氏温度到华氏温度的转换器。

```
(9 / 5) * temp + 32
```

如果temp为10，结果是什么？_____

这是一个布尔表达式。运算符==判断两个值是否相等。

```
color == "orange"
```

如果color的值为"pink"，这个表达式为true还是false？_____

如果color的值为"orange"呢？_____

```
name + ", " + "you've won!"
```

如果name的值为"Martha"，结果是什么？_____

这个运算符判断第一个值是否大于第二个值。你也可以使用>=来判断第一个值是否大于或等于第二个值。

```
yourLevel > 5
```

如果yourLevel的值为2，结果是什么？_____

如果yourLevel的值为5，结果是什么？_____

如果yourLevel的值为7，结果是什么？_____

```
(level * points) + bonus
```

假设level的值为5，points的值为30000，bonus的值为3300，结果是什么？_____

```
color != "orange"
```

如果color的值为"pink"，这个表达式为true还是false？_____

运算符!=判断两个值是否不相等。

选做题

```
1000 + "108"
```

会不会有多个答案呢？只有一个正确答案。你的答案是什么？_____



编码技巧

你注意到了吗？运算符=用于赋值，而运算符==用于判断相等性。也就是说，给变量赋值时使用一个等号，判断两个值是否相等时使用两个等号。一种常见的编码错误是，该使用其中一个运算符时使用了另一个。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

```
while (juggling) {
    keepBallsInAir();
}
```



重复操作

很多事情你都不只做一次：

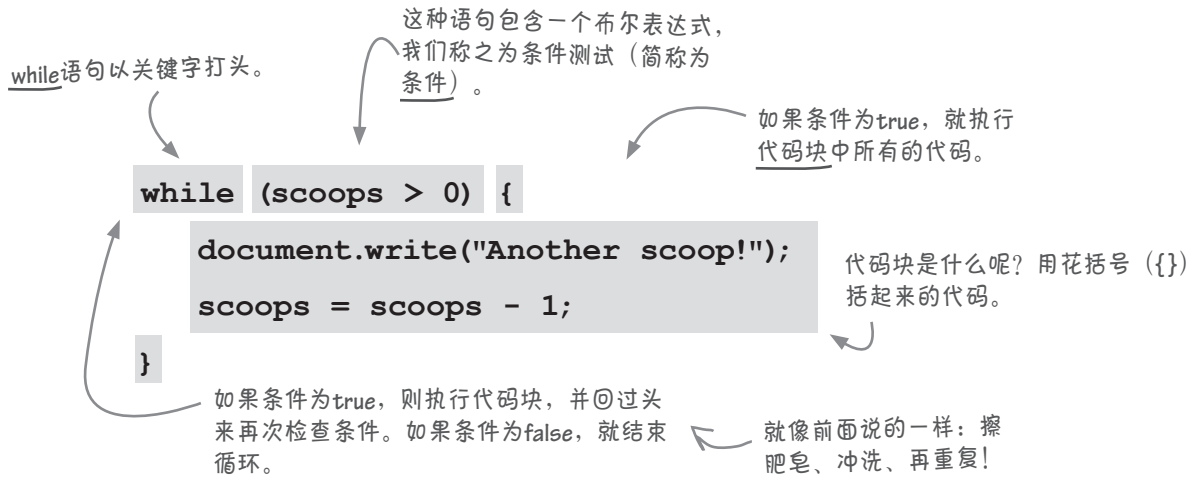
擦肥皂、冲洗、重复……

上蜡、刮蜡……

不停地吃，直到把碗里的糖果吃光。

你经常需要编写代码来重复执行相同的操作，而JavaScript提供了多种反复执行循环代码的方式：while、for、for in和forEach。本书将介绍所有这些循环方式，这里先将重点放在while循环上。

前面刚讨论过结果为布尔值的表达式，如scoops > 0。这种表达式在while语句中扮演着至关重要的角色，如下所示。



while循环的工作原理



假设这是你遇到的第一个while循环，下面来跟踪其整个执行过程，以了解其中的工作原理。注意到这里新增了一条语句，它声明变量scoops并将其初始化为5。

下面来执行这些代码。它首先将scoops设置为5。

```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```

接下来是while语句。执行while语句时，我们首先检查其中的条件，看看它是true还是false。

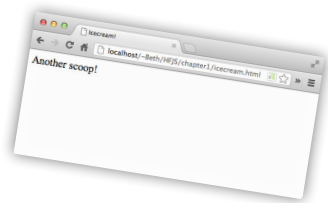
```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```

scoops大于0吗？在我们看来，它像是这样的！



由于条件为true，我们开始执行代码块，其中的第一条语句在浏览器中写入字符串"Another scoop!
"。

```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```



下一条语句将球数减1，并将结果（4）赋给scoops。




```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```

这是代码块中的最后一条语句，因此我们返回while语句开头并重新开始。

```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```

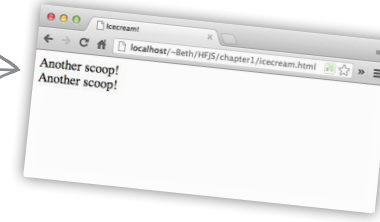
再次检查条件，这次scoops的值为4，依然大于0。




```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```

我们再次在浏览器中写入字符串"Another scoop!
"。

```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```



下一条语句将球数减1，并将结果（3）赋给scoops。

吃掉了2球，
还余下3球！



```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```

这是代码块中的最后一条语句，因此我们返回while语句开头并重新开始。

```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```

再次检查条件，这次scoops的值为3，依然大于0。

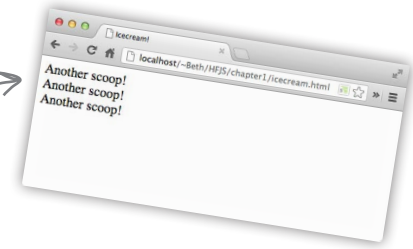
还余下很多球！



```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```

我们再次在浏览器中写入字符串"Another scoop!
"。

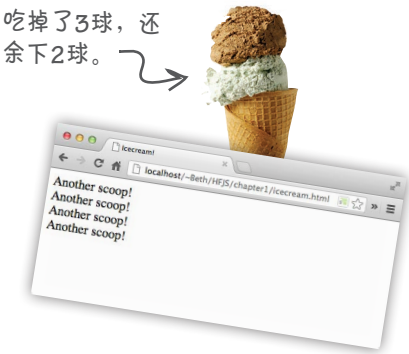
```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```



图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

正如你看到的，这个过程将不断重复。在每次循环中，我们都将scoops减1，再次将指定的字符串写入浏览器并重新开始。

```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```



继续执行循环。

```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```



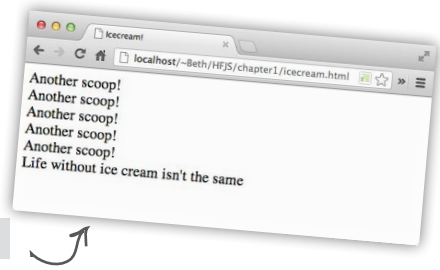
执行最后一次循环后，情况有点不同：scoops为0，因此条件为false。这次到达了分水岭，我们不再继续循环——不执行代码块。这次我们跳过代码块，执行它后面的语句。

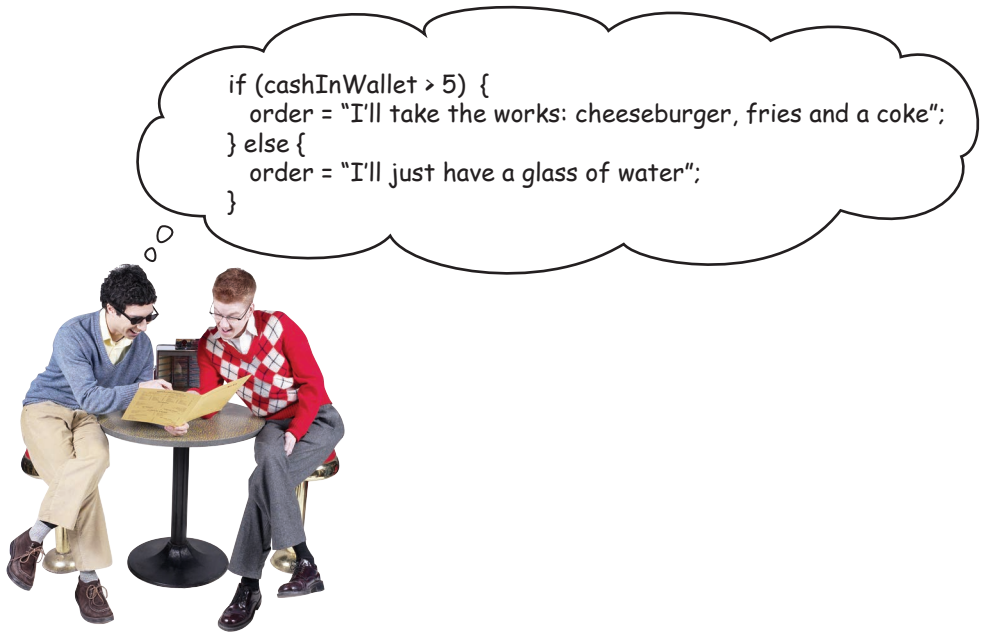
```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```



现在，我们执行另一条document.write语句，在浏览器中写入字符串"Life without ice cream isn't the same"。至此，全部代码都执行完毕了！

```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```





使用JavaScript进行决策

你刚才看到了，在while语句中，可使用条件来决定是否继续循环。在JavaScript中，还可在if语句中使用布尔表达式来进行决策。仅当if语句的条件测试为true时，才会执行其代码块，如下例所示。

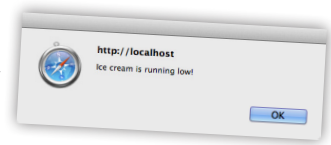
这是关键字if，后面是条件和代码块。

这个条件检查球数是否少于3。

```
if (scoops < 3) {
    alert("Ice cream is running low!");
}
```

如果少于3球，就执行if语句的代码块。

函数alert接受一个字符串，并在浏览器中显示一个包含该字符串的弹出式对话框。试一试吧！



在if语句中，还可添加一个或多个else if语句，以执行多重检查，如下所示：

使用if/else if语句，可依次进行两次检查。

```
if (scoops >= 5) {
    alert("Eat faster, the ice cream is going to melt!");
} else if (scoops < 3) {
    alert("Ice cream is running low!");
}
```

可根据需要添加任意数量的else if语句，以检查多个条件。每条else if语句都有其代码块，该代码块将在条件为true时执行。

进行大量决策

可根据需要将任意数量的if/else语句串接起来，还可在最后添加一条包罗万象的else语句，以处理所有条件都不满足的情形，如下所示。

```
if (scoops >= 5) {
   alert("Eat faster, the ice cream is going to melt!");
} else if (scoops == 3) {
   alert("Ice cream is running low!");
} else if (scoops == 2) {
   alert("Going once!");
} else if (scoops == 1) {
   alert("Going twice!");
} else if (scoops == 0) {
   alert("Gone!");
} else {
   alert("Still lots of ice cream left, come and get it.");
}
```

在这个代码段中，我们检查余下的球数是否不少于5……

……是否刚好为3……

……是否为2、1或0，并显示相应的提醒框。

如果上述条件都不满足，就执行这段代码。



世上没有愚蠢的问题

问：代码块到底是什么？

答：从语法上说，代码块是放在花括号内的一组语句，可以只包含一条语句，也可以包含任意数量的语句。代码块中的所有语句被视为一个整体，要么都执行，要么都不执行。例如，在while语句中，如果其条件为true，将执行其代码块中的所有语句。if和else if语句的代码块亦如此。

问：我遇到过这样的代码，其中的条件是一个变量，而这个变量的值为字符串而不是布尔值。请问这种代码是如何工作的？

答：本书稍后会讲到这一点；简单地说，JavaScript在判断值为true还是false方面非常灵活。例如，任何包含非空字符串的变量都被视为true，而没有设置值的变量都被视为false。这些细节稍后介绍。

问：你说过，表达式的结果可能不是数字、字符串或布尔值。请问还可能有哪些值？

答：我们当前的重点是基本类型，即数字、字符串和布尔值。本书后面将介绍更复杂的类型，如数组（一系列值）、对象和函数。

问：布尔值（boolean）是如何得名的？

答：布尔值是以发明布尔逻辑的英国数学家George Boole命名的。在英文中，boolean常被写作Boolean，旨在指出这种变量类型是以George Boole命名的。



代码冰箱贴

有一个JavaScript程序的代码被随机地贴在冰箱上。你能将这些冰箱贴放到正确的位置上，组成一个生成如下输出的JavaScript程序吗？请对照本章末尾的答案，再继续往下阅读。

正确地排列这些冰箱贴，组成一个能运行的JavaScript程序。

```
document.write("Happy Birthday dear " + name + ",<br>");

document.write("Happy Birthday to you.<br>");

var i = 0;

var name = "Joe";

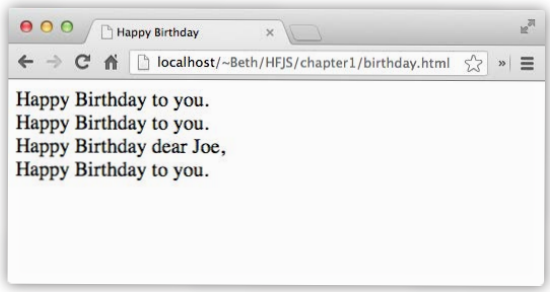
i = i + 1;

}

document.write("Happy Birthday to you.<br>");

while (i < 2) {
```

正确地排列这些冰箱贴后，组成的程序将生成如下输出：



请在这里正确地排列这些冰箱贴。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

与用户交流



前面一直在讨论如何提高网页的交互性，为此你必须能够与用户交流。实际上，与用户交流的方式有多种，其中的一些你在本章前面已经见到过。下面简要地介绍这些交流方式，本书后面将更详细地讨论它们。

本章将使用这三种交流方法。

创建提醒框

正如你已经看到的，浏览器提供了提醒用户的快捷方式——使用函数`alert`。你只需调用函数`alert`并指定一个包含提醒消息的字符串，浏览器就会在一个漂亮的对话框中显示这条消息。必须承认，由于`alert`易于使用，我们一直在滥用它；实际上，仅当你要停止一切并提醒用户时，才应使用它。

直接写入文档

将网页视为一个文档（浏览器就是这么认为的）。你随时都可使用函数`document.write`将任何HTML和内容写入网页，这虽然很常见，但通常被认为是一种糟糕的做法。本章也偶尔使用了这个函数，因为这是一种简单而容易掌握的网页操作方式。

使用控制台

所有JavaScript环境都包含控制台，可将代码中的消息写入其中。要将消息写入控制台日志，可使用函数`console.log`，并传入要写入的字符串（稍后将更详细地介绍如何使用控制台日志）。可将`console.log`视为杰出的故障排除工具，但用户通常看不到控制台日志，因此这并非与用户交流的有效方式。

在帮助找出代码错误方面，控制台真的非常方便！出现输入错误（如遗漏了引号）时，JavaScript通常会在控制台中显示错误消息，帮助你找出这种错误。

直接操作文档

这是最佳的方式，你应尽量使用这种方式来与网页和用户交互。使用JavaScript可以访问网页，读取和修改其内容，甚至修改其结构和样式！这些都是利用浏览器的**文档对象模型**（后面将更详细地讨论）实现的。你将看到，这是与用户交流的最佳方式。然而，要使用文档对象模型，你必须知道网页的结构，并熟悉用来读写网页的编程接口。这些都将在稍后介绍，在此之前，我们需要更深入地了解JavaScript。

这是我们的终极目标，届时你将能够以各种方式读取、修改和操作网页。



所有的交流方法都戴着面具来参加晚会了。你能揭下它们的面具，将右边的描述与左边的名称正确地搭配起来吗？我们已经将一种交流方法与其描述搭配起来了。

document.write

console.log

alert

文档对象模型

我让用户停止前进的步伐，并向他传递一条简短的消息。用户必须单击OK按钮才能继续前行。

我能够在文档中插入少量的HTML和文本。我虽然不是向用户传递消息的最佳方式，但在所有浏览器中都管用。

使用我可以全面控制网页：获取用户输入的值，修改HTML或样式，更新网页的内容。

我只是一种简单的调试工具。使用我可将信息写入开发人员专用的控制台。

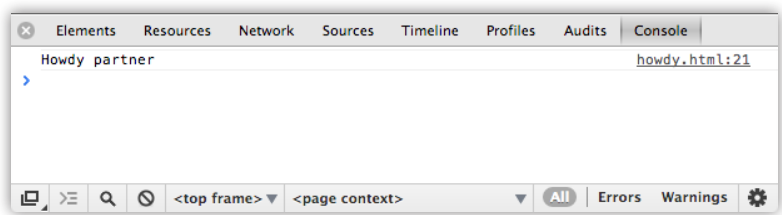
console.log详述

下面来更深入地探索console.log的工作原理，以便在本章中使用它来查看代码的输出，并在全书中使用它来查看代码的输出以及调试代码。但别忘了，控制台并非普通Web用户能够明白的浏览器功能，因此不应在网页的最终版本中使用它。将消息写入控制台日志通常仅用于在开发网页期间调试代码，但在学习JavaScript时，这是一种了解代码功能的极佳方式。console.log的工作原理如下：

获取任何字符串……

```
var message = "Howdy" + " " + "partner";
console.log(message);
```

……将其传递给*console.log*，该字符串将显示在浏览器的控制台中，如下所示。



控制台包含代码写入到日志的所有输出。

世上没有愚蠢的问题

问：我知道console.log可用于输出字符串，但它到底是什么东西呢？我是说为何使用句点分隔console和log呢？

答： 问得好。我们讲得有点太快了，你可将控制台视为具有特定功能的对象。其功能之一是写入日志，而要让控制台执行这种功能，我们使用语法console.log，并将用圆括号括起的输出传递给它。请牢记这一点，本书后面将回过头来深入讨论对象。现在，你具备了使用console.log所需的全部知识。

问：除了写入日志外，控制台还有其他功能吗？

答： 有，但大家通常只用它来写入日志。日志（和控制台）还有一些高级用法，但这些用法通常随浏览器而异。请注意，所有现代浏览器都提供了控制台，但控制台并不包含在正式规范中。

问：控制台看起来很不错，但在哪里能够找到它呢？我在代码中使用了它，却没有看到任何输出！

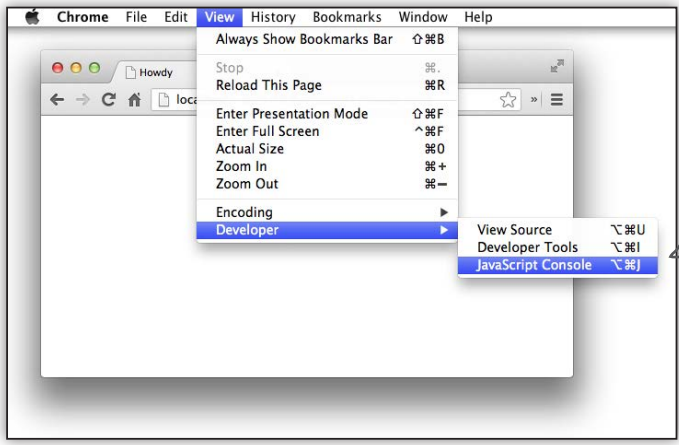
答： 在大多数浏览器中，都可显式地打开控制台，详情请参阅下一页。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

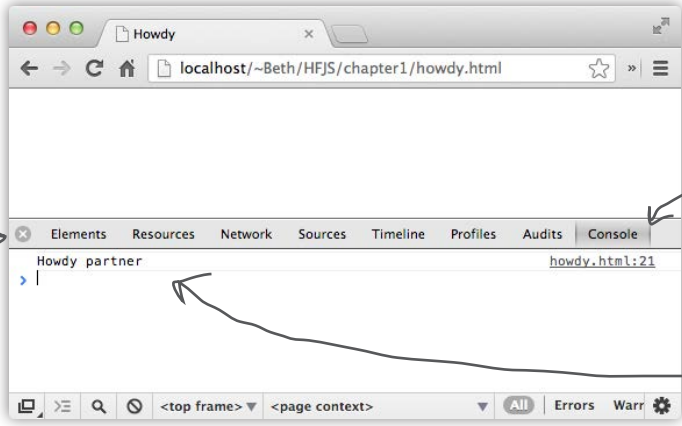
打开控制台

不同浏览器的控制台实现存在细微的差异，更糟糕的是，浏览器实现控制台的方式变化相当频繁——虽然没有达到离谱的程度，但当你阅读本书时，你的浏览器控制台可能与这里显示的稍有不同。

这里介绍在Mac上如何访问Chrome浏览器（25版）的控制台。对于如何在各种主流浏览器中访问控制台，请参阅http://wickedlysmart.com/hfjsconsole。明白如何在一款浏览器中访问控制台后，就很容易搞清楚如何在其他浏览器中访问控制台了。建议你至少在两款浏览器中尝试使用控制台，这样才更熟悉。



在Mac上，要访问Chrome的控制台，请选择菜单View > Developer > JavaScript Console。



控制台将出现在浏览器窗口的底部。

在控制台顶部的选项卡中，确保选择了标签Console。

在这个窗口中，你将看到在代码中使用console.log显示的所有消息。

不要考虑其他标签的作用。这些标签都很有用，但目前最重要的是Console，它能够让你看到在代码中使用console.log显示的所有消息。

编写一个正式的应用程序

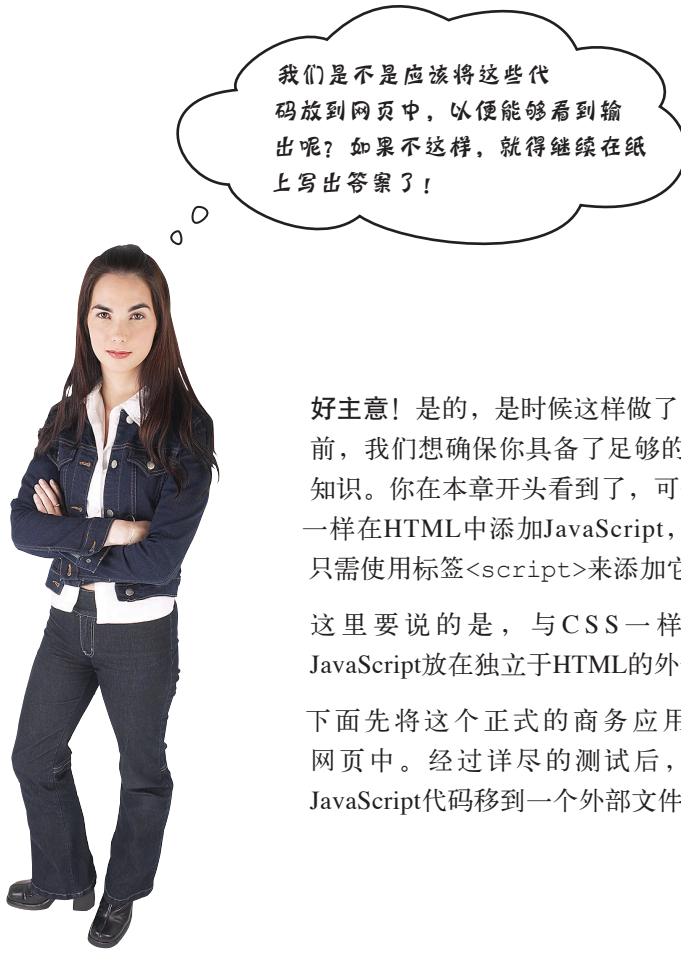


下面将新学到的JavaScript技能和`console.log`付诸应用，编写一个实用的应用程序。我们需要一些变量、一条`while`语句和一些带`else`子句的`if`语句。再进行简单打磨，不知不觉间就将得到一个正式的商务应用程序。查看最终代码前，先想想你会如何编写一个程序来显示经典歌曲*99 Bottles of Beer*的歌词。

```
var word = "bottles";
var count = 99;
while (count > 0) {
    console.log(count + " " + word + " of beer on the wall");
    console.log(count + " " + word + " of beer,");
    console.log("Take one down, pass it around,");
    count = count - 1;
    if (count > 0) {
        console.log(count + " " + word + " of beer on the wall.");
    } else {
        console.log("No more " + word + " of beer on the wall.");
    }
}
```



上述代码存在一个小缺陷：它能正确地运行，但输出并非完美无缺。你能找出并修复这个缺陷吗？



好主意！是的，是时候这样做了。这样做之前，我们想确保你具备了足够的JavaScript知识。你在本章开头看到了，可像添加CSS一样在HTML中添加JavaScript，也就是说，只需使用标签`<script>`来添加它们即可。

这里要说的是，与CSS一样，也可将JavaScript放在独立于HTML的外部文件中。

下面先将这个正式的商务应用程序放到网页中。经过详尽的测试后，再将这些JavaScript代码移到一个外部文件中。

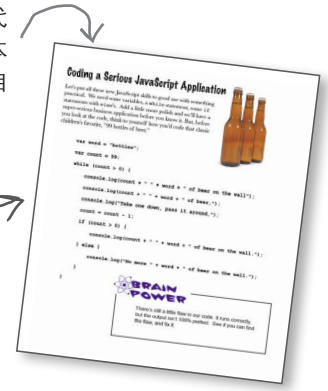


试驾

下面在浏览器中运行一些代码。请按如下说明来运行这个正式的商务应用程序，并查看最终的结果。

要下载本书的代码和示例文件，请访问http://wickedlysmart.com/hfjs。

1. 编写如下HTML代码，你将在其中嵌入JavaScript代码。首先输入这些HTML代码，再在<script>标签之间输入前面列出的JavaScript代码。你可使用记事本（Windows）或TextEdit（Mac）等编辑器，并确保处于纯文本模式；也可使用自己喜欢的HTML编辑器，如Dreamweaver、Coda或WebStorm。



输入这些HTML代码。

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>My First JavaScript</title>
  </head>
  <body>
    <script>

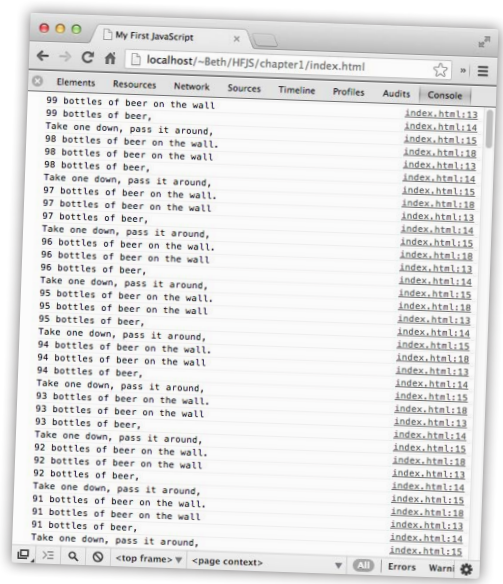
    </script>
  </body>
</html>
```

这些是<script>标签。现在你应该知道，必须将JavaScript代码放在这两个标签之间。

2. 将这个文件保存为index.html。

3. 在浏览器中加载这个文件。可以将这个文件拖放到浏览器窗口中，也可在浏览器中选择菜单“文件”>“打开”。

4. 在网页中，你什么都看不到，因为我们使用console.log将所有输出都写入了控制台。因此，请打开控制台，并祝贺自己编写了一个正式的商务应用程序。



这是代码的运行情况，它们生成歌曲*99 Bottles of Beer*的完整歌词，并将其写入浏览器的控制台。

如何将JavaScript代码加入网页（细数各种方式）

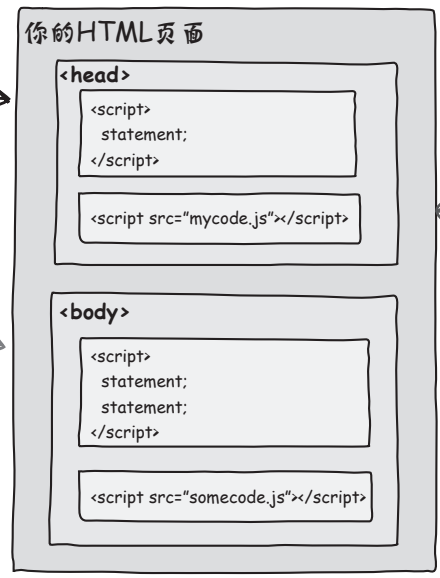
你已经知道可以使用<script>标签将JavaScript代码添加到网页的<head>或<body>元素中，但还有另外两种在网页中添加JavaScript代码的方式。下面来看看可将JavaScript代码放在哪些地方（以及将代码放在一个地方而不是另一个地方的原因）。

可将代码嵌入<head>元素中。在网页中添加代码时，最常见的方式是在<head>元素中添加一个<script>元素。这让你的代码更容易找到，好像也是放置代码的合理位置，但这并非总是最佳方式。为什么呢？请往下看。

也可以将代码内嵌在网页的<body>元素中。为此，可将JavaScript代码放在<script>元素中，并将<script>元素放到网页的<body>元素中（通常是最后）。

这种方法要好些。为什么呢？浏览器加载网页时，将先加载<head>元素内的所有内容，再加载<body>元素。因此，如果将代码放在<head>中，用户可能必须等一会儿才能看到网页。如果将代码放在<body>的HTML后面，则用户等待这些代码加载时就能看到网页的内容。

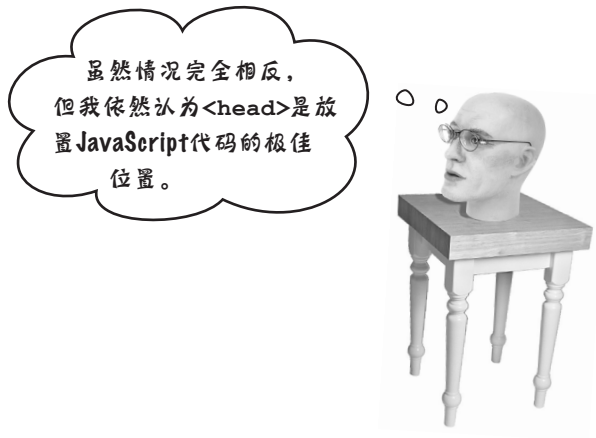
还有更好的方法吗？请接着往下看。



还可将代码放在独立的文件中，并在<head>中链接该文件。这与链接到CSS文件类似，唯一的差别是，你使用<script>标签的src特性来指定JavaScript文件的URL。

放在外部文件中，代码更容易维护（独立于HTML），还可用于多个网页。但这种方法也有缺点，那就是所有代码都将在网页的<body>部分之前加载。还有更佳的方式吗？请继续往下看。

最后，可在网页的<body>元素中链接到外部文件。这是鱼和熊掌兼得的最佳方式，既有一个可用于任何网页且易于维护的JavaScript文件，又在网页末尾引用它，这样它将在网页加载完毕后再加载。真不错。



棒打鸳鸯散

分手是痛苦的，但必须这样做。现在该将JavaScript代码提取出来，移到一个独立的文件中了，具体步骤如下。

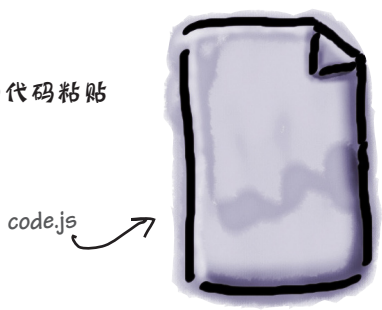
1. 打开文件index.html，选择其中所有的JavaScript代码，即标签<script>之间的所有内容，如下所示。

```
<!doctype html>
<html lang="en">

  <head>
    <meta charset="utf-8">
    <title>My First JavaScript</title>
  </head>
  <body>
    <script>
      var word = "bottles";
      var count = 99;
      while (count > 0) {
         console.log(count + " " + word + " of beer on the wall");
         console.log(count + " " + word + " of beer,");
         console.log("Take one down, pass it around,");
         count = count - 1;
         if (count > 0) {
            console.log(count + " " + word + " of beer on the wall.");
         } else {
            console.log("No more " + word + " of beer on the wall.");
         }
      }
    </script>
  </body>
</html>
```

只选择JavaScript代码，不要包含<script>标签，因为在独立文件中不需要这些标签。

2. 在编辑器中新建一个文件，将其命名为code.js，并将前面复制的代码粘贴到其中，然后保存。



③ 现在需要在文件index.html中添加一个指向文件code.js的引用，以便该网页加载时获取并加载文件code.js。为此，将文件index.html中的JavaScript代码删除，但保留<script>标签；再在<script>开始标签中添加一个引用code.js的src特性。

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>My First JavaScript</title>
  </head>
  <body>
    <script src="code.js">

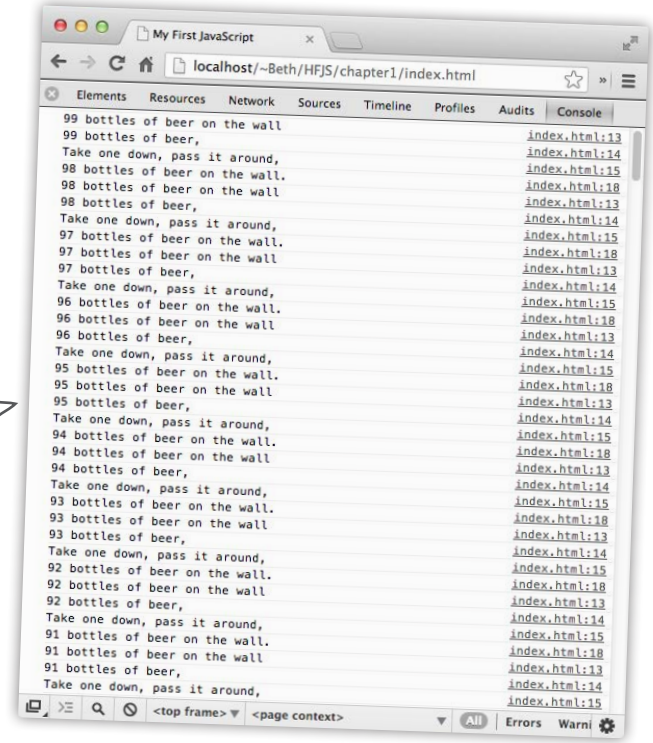
    </script>
  </body>
</html>
```

使用<script>元素的src特性链接到JavaScript文件。

这是被删除的代码所处的位置。

信不信由你，<script>结束标签依然必不可少，虽然这两个标签之间已经没有代码了。

④ 外科手术到此结束，就这么简单。现在需要进行测试了。为此，再次加载index.html，结果应该与前面完全相同。请注意，这里将src特性设置成了"code.js"，即假定这个代码文件与HTML文件位于同一个目录。



结果应与前面相同。但现在HTML和JavaScript位于不同的文件中，是不是更清晰、更容易管理、不那么局促了呢？

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权



剖析script元素

你已经知道如何使用<script>元素在网页中添加代码了，但为了彻底明白这个主题，下面来复习一下<script>元素，确保不放过任何一个细节。

特性type告诉浏览器，你要编写JavaScript代码。如果省略这个属性，浏览器将假定你使用的是JavaScript。鉴于此，我们建议你省略这个属性，标准制定者也是这样建议的。

<script起始标签。

```
<script type="text/javascript">
  alert("Hello world!");
</script>
```

别忘了起始标签中的右尖括号。

<script>标签之间的所有内容都必须是有效的JavaScript代码。

在任何情况下，</script>结束标签都必不可少。

在HTML中引用独立的JavaScript文件时，像下面这样使用<script>元素：

添加一个src特性来指定JavaScript文件的URL。

```
<script src="myJavaScript.js">
</script>
```

JavaScript文件使用扩展名.js。

千万别忘了</script>结束标签！即便是链接外部文件时，这个标签也必不可少。

引用独立的JavaScript文件时，<script>元素不能包含任何JavaScript代码。



不能在引用JavaScript文件的同时内嵌JavaScript代码。

使用src特性时，不能在<script>标签之间添加JavaScript代码。在这种情况下，需要使用两个<script>元素。

```
<script src="goodies.js">
  var = "quick hack";
</script>
```

错误的做法



起底JavaScript

本周访谈：逐渐了解JavaScript

Head First：欢迎JavaScript。我们知道你始终忙于处理各种网页，很高兴你在百忙之中抽出时间接受访谈。

JavaScript：没什么。我的确比以前更忙了，现在我不仅被用于几乎所有的网页，还被用于实现简单的菜单效果和功能齐备的游戏等。真是忙得不可开交！

Head First：真令人惊讶。就在几年前，还有人说你不过是一款无用的脚本语言，而现在到处都能看到你的身影。

JavaScript：以前的事就别说了。我取得了长足的进步，很多杰出人物都为此付出了艰苦努力。

Head First：取得了什么进步呢？你的基本语言特性看起来不是跟以前一样吗？

JavaScript：这表现在两个方面。首先，现在我的速度快如闪电。我虽然被视为脚本语言，但性能几乎能够与编译型语言媲美。

Head First：其次呢？

JavaScript：在浏览器中，我能够做的事情多得多了。通过使用所有现代浏览器都有的JavaScript库，你可确定当前位置，播放视频和音频，在网页中绘图，等等。但要完成这些任务，你必须熟悉JavaScript。

Head First：咱们来说说对你的批评吧。我听过一些不那么友好的评论，其中最著名的是“一款粗制滥造的语言”。

JavaScript：走自己的路，让别人去说吧。我即便不是世界上使用最广泛的语言，也是其中之一。我击败了很多竞争对手。还记得将Java用于浏览器的倡导吗？现在看来就是个笑话。还有VBScript、JScript、Flash、Silverlight……不胜枚举。你说说，我怎么就不行呢？

Head First：有人批评你“过分简单”。

JavaScript：坦率地说，这是我最大的优点。事实上，只要启动浏览器，输入几行JavaScript代码，就万事大吉了，这多厉害呀。对初学者来说，这也很不错。有人说没有比JavaScript更好的入门语言了。

Head First：但这也是需要付出代价的，不是吗？

JavaScript：问得好。我简单是从易于上手的意义上说的。我也很深奥，最新的现代编程结构应有尽有。

Head First：比方说呢？

JavaScript：动态类型、一等函数和闭包，算吗？

Head First：都算，不过它们都是什么东西呢？

JavaScript：只要坚持阅读本书，你就会知道。

Head First：说点具体的吧。

JavaScript：我就说一点。JavaScript是针对动态Web环境打造的，在这种激动人心的环境中，用户与网页交互，数据是即时获得的，还会发生各种各样的事件，而JavaScript反映了这种编程风格。对JavaScript有更深入的了解后，你就能更好地理解这一点。

Head First：听你这样说，你是完美的语言了？

JavaScript（**泪流满面**）：你知道，我不像其他大多数语言那样出生在象牙塔。我出身草莽，成败全靠自己，必须努力拼搏。虽然如此，我并非完美无缺，也有一些“缺点”。

Head First（**面带Barbara Walters[1]式微笑**）：我们今天看到了你不为人知的一面，我想改天有必要再来一次访谈。离别之际，有什么要说的吗？

JavaScript：不要只看我的缺点，要了解并坚持利用我的优点。

① Barbara Walters是美国著名访谈类节目主持人，被誉为“美国电视新闻第一夫人”。——编者注



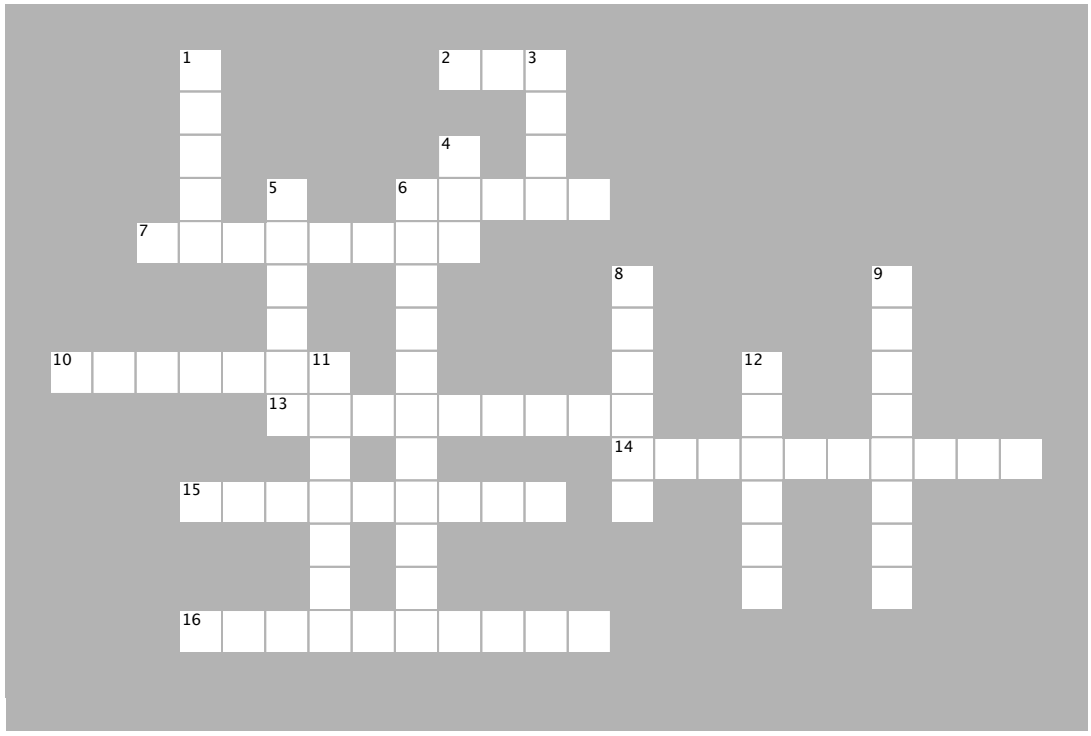
要点

- JavaScript用于给网页添加**行为**。
- 与几年前相比，浏览器引擎执行JavaScript代码的速度快得多。
- 浏览器在网页中遇到JavaScript代码后就开始执行它们。
- 使用`<script>`元素在网页中添加JavaScript代码。
- 可在网页中内嵌JavaScript代码，也可在HTML中链接到包含JavaScript代码的独立文件。
- 要链接到独立的JavaScript文件，可在`<script>`标签中使用特性`src`。
- HTML**声明**网页的结构和内容；JavaScript**计算**值并给网页添加行为。
- JavaScript程序由一系列**语句**组成。
- 变量声明语句是最常见的JavaScript语句之一，它使用关键字`var`来声明新变量，并使用赋值运算符=给变量赋值。
- JavaScript变量的命名规则和指南只有几条，遵循它们至关重要。
- 给变量命名时千万不要使用关键字。
- JavaScript表达式计算值。
- 三种常见的表达式是**数字表达式**、**字符串表达式**和**布尔表达式**。
- `if/else`语句让你能够在代码中作出决策。
- `while/for`语句让你能够通过循环多次执行代码。
- 使用`console.log`（而不是`alert`）将消息显示到控制台。
- 控制台消息应该只用于调试，因为用户很可能根本看不到控制台消息。
- JavaScript最常用于给网页添加行为，但也用于在Adobe Photoshop、OpenOffice和Google Apps等应用程序中编写脚本，甚至被用作服务器端编程语言。



JavaScript填字游戏

来玩一个填字游戏[①]，放松一下心情，让你的神经完全处于松弛状态。



横向

2. 在HTML中链接到外部JavaScript文件时，需要在<script>元素中指定的特性。

6. 为避免难堪的命名错误而使用的拼写方式。

7. JavaScript给网页添加的东西。

10. 前面例子中用到的啤酒的计量单位（复数形式）。

13. 表示一行JavaScript代码。

14. 3 + 4所属的JavaScript语言结构。

15. 所有JavaScript语句都以它结束。

16. 用于调试JavaScript代码的函数。

纵向

1. 一种用于在JavaScript程序中重复执行操作的循环。

3. JavaScript变量名是区分什么的?

4. 声明变量时使用的关键字。

5. 变量存储的东西。

6. 每次执行循环时，都要判断的一个表达式。

8. 当今JavaScript的运行速度比以前更快还是更慢?

9. if/else语句的用途。

11. 可使用+运算符将什么拼接起来?

12. 用于放置JavaScript代码的元素。

① 本书中的填字游戏均为填写英语单词。——编者注

变身浏览器答案



下面的JavaScript代码存在一些错误。你的任务是变身浏览器，将其中的错误找出来。完成这个练习后，请翻到本章末尾，看看你是否找出了所有的错误。答案如下。

A

```
// 笑话测试
var joke = "JavaScript walked into a bar....';
var toldJoke = "false";
var $punchline =
  "Better watch out for those semi-colons."
var %entage = 20;
var result

if (toldJoke == true) {
    Alert($punchline);
} else
    alert(joke);
}
```

将字符串括起时，要么使用两个双引号（"），要么使用两个单引号（'）。不要混用。

除非需要指定的是字符串，否则不要用引号将布尔值括起。

可以使用以$打头的变量名，但不建议这样做。

别忘了在语句末尾添加分号！

变量名中不能使用%。

也是在末尾遗漏了分号。

JavaScript是区分大小写的，应为alert而不是Alert。

遗漏了一个左花括号。

B

```
\\ 电影之夜
var zip code = 98104;
var joe'sFavoriteMovie = Forbidden Planet;
var movieTicket$    =    9;

if (movieTicket$ >= 9) {
    alert("Too much!");
} else {
    alert("We're going to see " + joe'sFavoriteMovie);
}
```

注释应以//而不是\\打头。

变量名不能包含空格。

变量名不能包含引号。

但指定字符串Forbidden Planet时，必须用引号括起。

由于变量名非法，这条if/else语句不管用。



磨笔上阵答案

请拿起笔来，将一些表达式的作用搞清楚。计算下面每个表达式的值并写下来。是的，写下来。将老妈对你不要在书上乱写乱画的嘱咐忘了吧，在本书中想怎么写就怎么写！一定对照本章末尾的答案，看看你的答案对不对。答案如下。

这是一个摄氏温度到华氏温度的转换器。

```
(9 / 5) * temp + 32
```

如果temp为10，结果是什么？ 50

这是一个布尔表达式。运算符==判断两个值是否相等。

```
color == "orange"
```

如果color的值为"pink"，这个表达式为true还是false？ false
如果color的值为"orange"呢？ true

```
name + ", " + "you've won!"
```

如果name的值为"Martha"，结果是什么？
“Martha, you've won!”

这个运算符判断第一个值是否大于第二个值。你也可以使用>=来判断第一个值是否大于或等于第二个值。

```
yourLevel > 5
```

如果yourLevel的值为2，结果是什么？ false
如果yourLevel的值为5，结果是什么？ false
如果yourLevel的值为7，结果是什么？ true

```
(level * points) + bonus
```

假设level的值为5，points的值为30000，bonus的值为3300，结果是什么？ 153300

```
color != "orange"
```

运算符!=判断两个值是否不相等。

如果color的值为"pink"，这个表达式为true还是false？ true

选做题

```
1000 + "108"
```

会不会有多个答案呢？只有一个正确答案。你的答案是什么？
“1000108”



编码技巧

你注意到了吗？运算符=用于赋值，而运算符==用于判断相等性。也就是说，给变量赋值时使用一个等号，判断两个值是否相等时使用两个等号。一种常见的编码错误是，该使用其中一个运算符时使用了另一个。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权



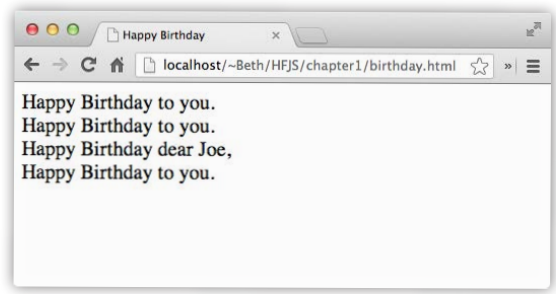
代码冰箱贴答案

有一个JavaScript程序的代码被随机地贴在冰箱上。你能将这些冰箱贴放到正确的位置上，组成一个生成如下输出的JavaScript程序吗？请对照本章末尾的答案，再继续往下阅读。答案如下。

正确排列后的冰箱贴！

```
var name = "Joe";
var i = 0;
while (i < 2) {
    document.write("Happy Birthday to you.<br>");
    i = i + 1;
}
document.write("Happy Birthday dear " + name + ",<br>");
document.write("Happy Birthday to you.<br>");
```

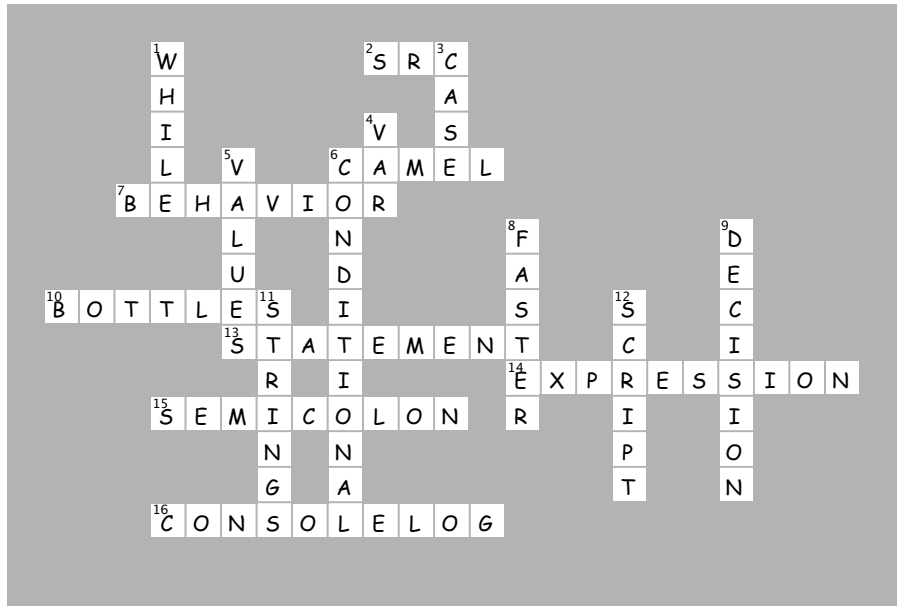
正确地排列这些冰箱贴后，组成的程序将生成如下输出。



图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权



JavaScript 填字游戏答案



连连看答案

所有的交流方法都戴着面具来参加晚会了。你能揭下它们的面具，将右边的描述与左边的名称正确地搭配起来吗？我们已经将一种交流方法与其描述搭配起来了。答案如下。

document.write

console.log

alert

文档对象模型

我让用户停止前进的步伐，并向他传递一条简短的消息。用户必须单击OK按钮才能继续前行。

我能够在文档中插入少量的HTML和文本。我虽然不是向用户传递消息的最佳方式，但在所有浏览器中都管用。

使用我可以全面控制网页：获取用户输入的值，修改HTML或样式，更新网页的内容。

我只是一种简单的调试工具。使用我可将信息写入开发人员专用的控制台。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

2 编写代码

更进一步



你已经知道了变量、类型、表达式等，我们接着往下介绍。你对JavaScript有所了解，实际上已经具备了足够的知识，可以编写一些**真正的代码**了：一些完成有趣功能的代码，一些有人想使用的代码。你缺乏的是实际编写代码的**经验**，现在就来弥补吧。如何弥补呢？全身心地投入，开发一个完全使用JavaScript编写的休闲类游戏。我们目标远大，又脚踏实地，一步一个脚印。来吧，现在就开始。如果你想利用这款游戏来一次创业，我们不会有任何意见，因为代码是你的。

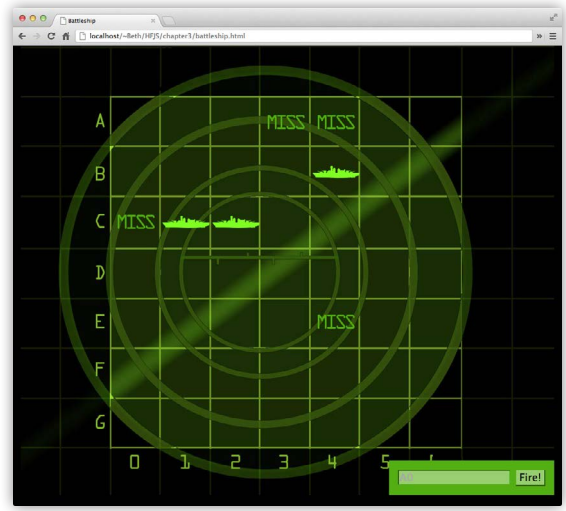
开发一款战舰游戏

你将充当浏览器的对手：浏览器将战舰隐藏起来，你的任务是找到并击沉这些战舰。当然，不同于真正的战舰游戏，在这款游戏中，你并没有自己的战舰。你的目标是通过尽可能少的猜测次数将浏览器的战舰全部击沉。

目标：通过尽可能少的猜测次数将浏览器的战舰全部击沉。将根据你的表现给你打分。

准备工作：这个游戏启动时，计算机将战舰放在一个虚拟网格中。然后，游戏将让你进行第一次猜测。

玩法：浏览器将提示你猜测位置，然后你输入网格位置。你猜测后，将显示结果Hit、Miss或You sank my battleship!。所有战舰都被击沉后，游戏结束并显示你的得分。



我们的努力目标如下：一个漂亮的7×7网格，其中包含三艘目标战舰。我们将从更简单的版本着手，等你对JavaScript有更深入的了解后，再添加图形和其他内容，让游戏与这里显示的完全一样。我们将把添加音效的任务留给你去完成。

第一次尝试——简化的战舰游戏

完整版使用7×7网格，有三艘战舰，但这里将编写一个更简单的版本：使用1×7网格，只有一艘战舰。这有点粗糙，但我们的重点是设计游戏的基本代码，而不是满足感官，至少现在如此。

不用担心，通过从这个游戏的简化版着手，将为后面编写完整版打下坚实的基础。这也提供了足够多的难题，让我们有机会在第一个真正的JavaScript程序（当然，第1章那个正式的商务应用程序不算）中解决。因此，本章将创建这个游戏的简化版，等你学习到更多的JavaScript知识后再创建其豪华版。

不使用上面所示的7×7网格，我们将从1×7网格着手。另外，我们暂时只考虑一艘战舰。



请注意，每艘战舰都占据三个单元格。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

从高层次设计着手

我们知道，我们需要一些变量、数字和字符串，还需要if语句、条件测试和循环等，但在什么地方需要、需要多少呢？如何将这些整合起来呢？要回答这些问题，我们需要更详细地了解这款游戏的功能。

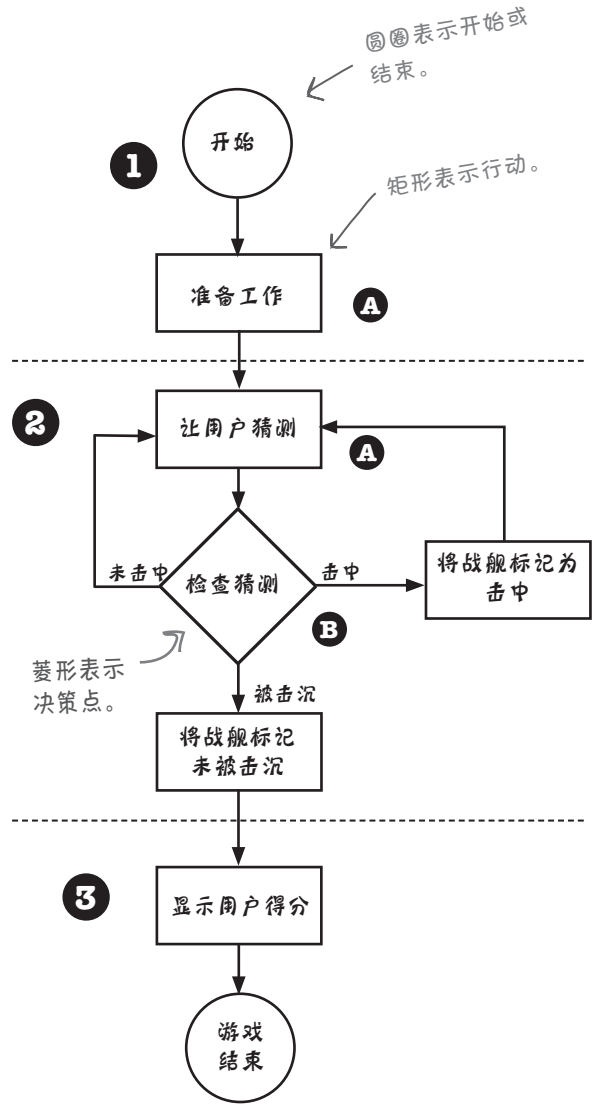
首先，需要清楚这款游戏的大致流程。基本情况如下。

1. 用户启动游戏
   - A 游戏将一艘战舰随机地放在网格中。
2. 开始玩游戏

   重复下面的步骤，直到战舰被击沉。
   - A 提示用户猜测一个位置（2、0等）
   - B 将用户猜测的位置与战舰的位置进行比较，确定结果是击中、未击中还是击沉。
3. 游戏结束

   根据猜测次数给用户打分。

对游戏需要完成的任务有大致了解后，接下来需要了解这些步骤的更多细节。



哇，一个货真价实的流程图。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

更多细节

我们有了高层次设计和专业级流程图，对这款游戏的工作原理有了非常深入的认识，但开始编码前，还需要厘清其他一些细节。

表示战舰

首先，需要确定如何表示网格中的战舰。别忘了，这个网格是**虚拟**的，它并不存在于这个程序中。只要用户和游戏知道战舰隐藏在网格中3个连续的单元格内就行（单元格的编号从零开始），在代码中并不需要将网格本身表示出来。你可能想创建存储全部7个单元格的东西，再将战舰放置到这些位置。但实际上不需要这样做，只需知道战舰在哪些单元格（如单元格1～3）中即可。

获取用户输入

如何获取用户输入呢？可使用`prompt`函数。每当需要让用户猜测位置时，我们都使用`prompt`来显示一条消息，并从用户那里获取输入（一个0～6的数字）。

显示结果

如何显示输出呢？就现在而言，我们将继续使用`alert`来显示游戏的输出。这虽然略显简陋，但确实管用（本书后面编写真正的游戏时，将更新网页，但在此之前，我们还有一段路要走）。

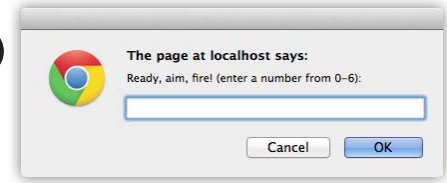
1 **游戏启动**。创建一艘战舰，并指定它在排列成一行的7个单元格中占据哪三个单元格。

这些单元格是使用整数表示的。例如，1、2和3表示战舰占据了如下图所示的单元格：



2 **游戏开始**。提示用户进行猜测。

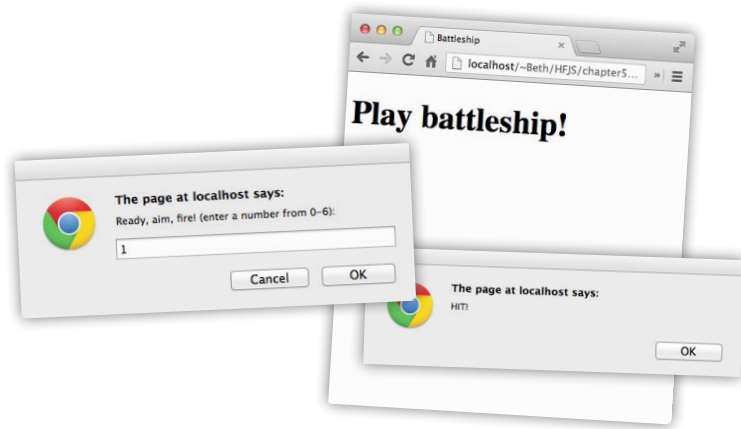
A



B 检查用户的输入是否是战舰占据的三个单元格之一。在一个变量中记录击中次数。

3 战舰的3个位置都被击中且击中次数为3时**游戏结束**。告诉用户他猜测了多少次才将战舰击沉。

游戏与用户之间的交互



编写伪代码

我们需要一种规划和编写代码的方法。首先来编写**伪代码**（pseudocode）。伪代码介于JavaScript代码和程序的自然语言描述之间。你将看到，它让我们无需编写**实际代码**就能彻底弄清楚程序的运行过程。

下面的伪代码由两部分组成，第一部分描述了我们需要的变量，而第二部分描述了程序的逻辑。变量指出了要在代码中记录哪些东西，而逻辑指出了要创建这个游戏必须如实地实现哪些代码。

我们需要的变量

声明三个**变量**，用于存储战舰占据的各个单元格。将这些变量命名为location1、location2和location3。

声明一个存储用户猜测的**变量**，将其命名为guess。

声明一个存储击中次数的**变量**，将其命名为hits并**初始化**为0。

声明一个存储猜测次数的**变量**，将其命名为guesses并**初始化**为0。

声明一个记录战舰是否被击沉的**变量**，将其命名为isSunk并**初始化**为false。

下面是游戏的逻辑。

循环：只要战舰未被击沉

  获取用户的猜测

  将用户输入与有效的输入值进行**比较**

  如果用户的猜测无效

    就**显示**让用户输入有效的数字

  否则

    将猜测次数**加**1

    如果用户猜测的是战舰占据的位置之一

      就将击中次数**加**1

      如果击中次数为3

        就将isSunk**设置**为true

        并向用户**显示**You sank my battleship!

      如果到此结束

    如果到此结束

  否则到此结束

循环到此结束

向用户**显示**统计信息

这不是JavaScript代码，但你可能已经知道如何使用代码实现这种逻辑了。

注意，这里使用了缩进让伪代码更容易理解。在实际代码中，我们也会这样做。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

磨笔上阵

假设我们的虚拟网格类似于下面这样：



并像下面这样使用了位置变量来指出战舰的位置：

```
location1 = 3;
location2 = 4;
location3 = 5;
```

而用户的输入序列如下：

```
1, 4, 2, 3, 5
```

在给定上述用户输入的情况下，前一页的伪代码将如何执行呢？请将你认为的结果写在下面。我们指出了开始时的情况。如果这是你第一次演练伪代码的运行过程，请花点时间彻底弄清楚其工作原理。

如果你需要帮助，可以偷看一眼本章末尾的答案。

location1	location2	location3	guess	guesses	hits	isSunk
3	4	5	——	0	0	false
3	4	5	1	1	0	false

第一行列出了用户第一次猜测之前各个变量的值。我们没有初始化变量guess，因此它的值是未定义的。

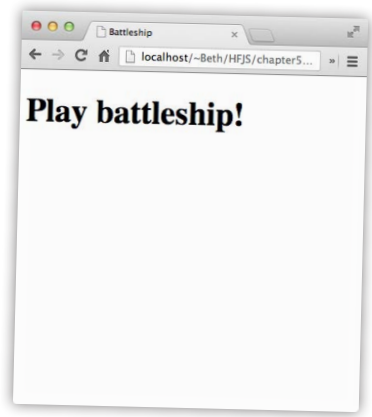
对了，别忘了HTML

如果没有链接到JavaScript代码的HTML，你就走不了多远。新建一个文件，将其命名为battleship.html，并输入下面的标记。等你编写好这个HTML文件后，我们将回过头去编写JavaScript代码。

战舰游戏的HTML超级简单。我们只是需要一个链接到JavaScript代码的网页，而所有的操作都是在JavaScript代码中完成的。

```
<!doctype html>
<html lang="en">
  <head>
    <title>Battleship</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Play battleship!</h1>
    <script src="battleship.js"></script>
  </body>
</html>
```

我们在这个网页的<body>元素末尾链接到JavaScript代码，因此等浏览器开始执行battleship.js中的代码时，网页已加载完毕。



这个网页刚加载完毕时的结果。我们需要编写一些实现游戏的代码！

考考你的脑力

集中精神。

虽然有点超前，但请你思考下列问题：加载网页时，为随机选择战舰占据的单元格，你认为需要什么样的代码呢？为正确地放置战舰，需要考虑哪些因素呢？将你的想法写在下面。

编写简单的战舰代码

我们将以伪代码为蓝本，编写实际的JavaScript代码。首先，来处理需要的所有变量。再来看一眼伪代码，搞清楚需要的变量。

我们需要三个变量来存储战舰的位置。

声明三个**变量**，用于存储战舰占据的各个单元格。将这些变量命名为location1、location2和location3。

还需要另外三个变量（guess、hits和guesses）来处理用户的猜测。

声明一个存储用户猜测的**变量**，将其命名为guess。

声明一个存储击中次数的**变量**，将其命名为hits并*初始化*为0。

声明一个存储猜测次数的**变量**，将其命名为guesses并*初始化*为0。

还需要一个变量来记录战舰是否被击沉。

声明一个记录战舰是否被击沉的**变量**，将其命名为isSunk并*初始化*为false。

下面在一个JavaScript文件中创建这些变量。新建一个文件，将其命名为battleship.js，并输入如下变量声明。

这是三个位置变量。我们暂时将战舰的位置设置为3、4和5。

后面将回过头来编写一些代码，它们生成随机的战舰位置，让用户更难找到。

```
var location1 = 3;
var location2 = 4;
var location3 = 5;
```

用户猜测前，变量guess没有值，因此是未定义的。

```
var guess;
var hits = 0;
var guesses = 0;
```

我们将变量hits和guesses的初始值都设置为0。

```
var isSunk = false;
```

最后，设置变量isSunk的初始值为false。用户击沉战舰后，我们将把这个变量设置为true。



编码技巧

如果你没有给变量指定初始值，JavaScript将给它指定默认值undefined。JavaScript使用undefined来指出还没有给变量赋值。本书后面将更详细地讨论undefined和其他古怪的值。

编写游戏逻辑

声明好变量后，下面来深入挖掘实现游戏逻辑的伪代码。我们将把这些伪代码分为几部分。你首先要做的是实现循环：它需要不断循环，直到战舰被击沉；接下来，获取用户的猜测并验证它是否有效，即确保它是0～6的数字；然后，编写检查战舰是否被击中以及是否被击沉的逻辑；最后，向用户报告，指出他猜测了多少次才将战舰击沉。



循环：只要战舰未被击沉

第1步：创建循环，获取并验证用户输入

    获取用户的猜测

    将用户输入与有效的输入值进行**比较**

    如果用户的猜测无效

        就**显示**让用户输入有效的数字

    否则

第2步：检查猜测的位置，确定有没有击中战舰

        将猜测次数**加**1

        如果用户猜测的是战舰占据的位置之一

            就将击中次数**加**1

第3步：检查战舰是否被击沉

            如果击中次数为3

                就将`isSunk`**设置**为`true`

                并向用户**显示**You sank my battleship!

            如果到此结束

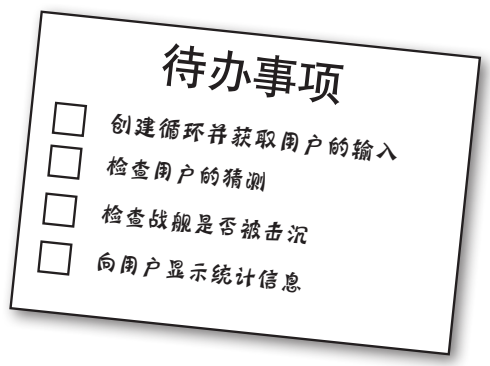
        如果到此结束

    如果到此结束

循环到此结束

第4步：处理要向用户显示的最终消息

向用户**显示**统计信息



第一步：创建循环并获取输入

- [ ] 创建循环并获取用户的输入
- [ ] 检查用户的猜测
- [ ] 检查战舰是否被击沉
- [ ] 向用户显示统计信息

下面来将游戏逻辑转换为实际的JavaScript代码。JavaScript代码和伪代码之间并不是完全对应的，因此在有些地方必须进行调整。伪代码让我们很清楚需要做什么，现在我们必须编写JavaScript代码，以解决如何做的问题。

先来看看到目前为止已编写的全部代码，然后将注意力聚焦于要添加的代码上（这样可节省纸张；如果你阅读的是电子版，节省的就是电能）。

这些已经介绍过，这里包含它们只是出于完整性考虑。

声明变量

```
var location1 = 3;
var location2 = 4;
var location3 = 5;
var guess;
var hits = 0;
var guesses = 0;
var isSunk = false;
```

循环从这里开始。只要战舰未被击沉，游戏就不会结束，因此继续循环。

前面说过，while语句使用条件测试来决定是否继续循环。在这里，我们通过检查确认isSunk仍为false。一旦战舰被击沉，就会将isSunk设置为true。

循环：只要战舰未被击沉

```
while (isSunk == false) {
```

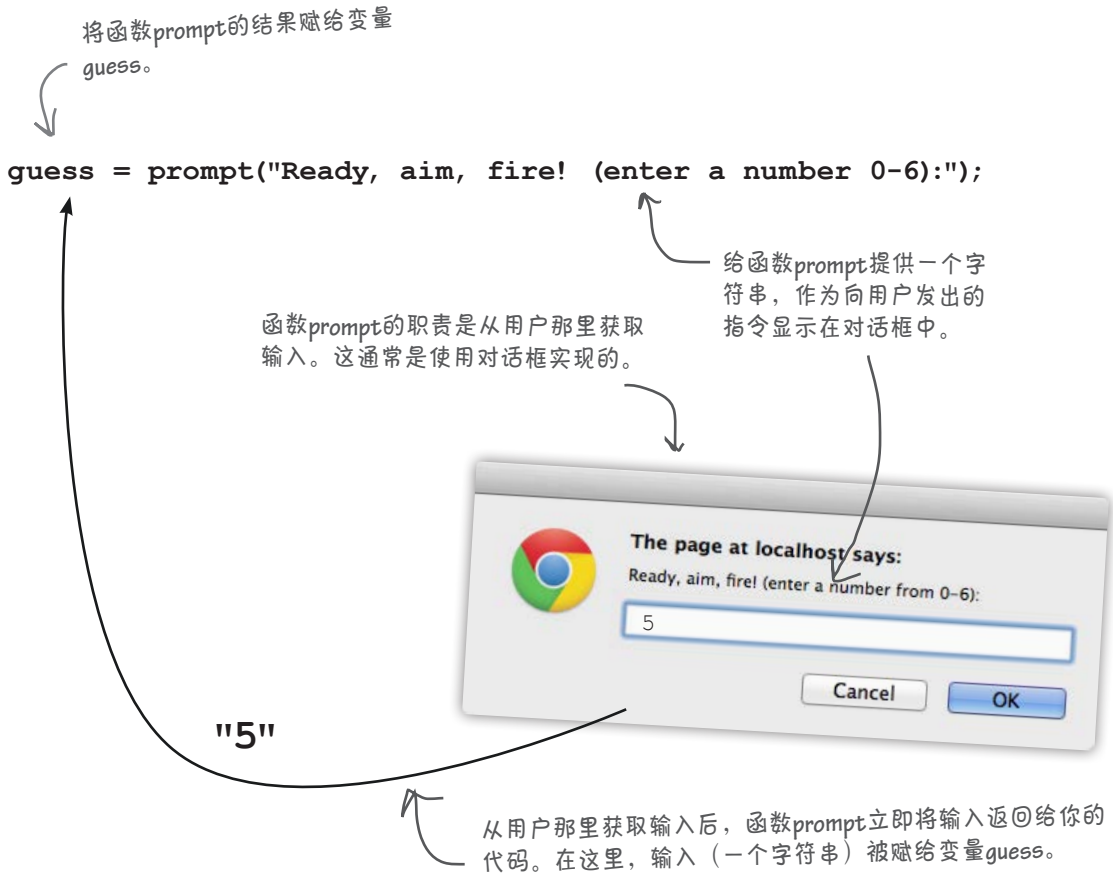
获取用户的猜测

```
    guess = prompt("Ready, aim, fire! (enter a number 0-6):");
}
```

每次循环时，都让用户进行猜测。为此，我们使用了内置函数prompt，下一页将更详细地介绍这个函数。

prompt的工作原理

浏览器提供了一个内置函数，可以用来获取用户输入；这个函数就是prompt。函数prompt与你使用过的函数alert很像（它也显示一个对话框，其中包含你指定的字符串），但它还提供了让用户输入响应的区域。这种响应将作为函数调用结果以字符串的形式返回；如果用户取消了对话框或没有输入任何响应，返回的将是null。





你现在可能想尝试运行这些代码，但千万不要这样做。

如果你这样做，浏览器将开始一个**无限循环**，不断地让你猜测。除非让操作系统强行停止浏览器进程，否则根本没法停止这个循环。

检查用户的猜测

- [x] 创建循环并获取用户的输入
- [ ] 检查用户的猜测
- [ ] 检查战舰是否被击沉
- [ ] 向用户显示统计信息

从伪代码可知，要检查用户的猜测，首先需要确认用户的输入有效；如果有效，就检查是否击中了战舰。我们还需相应地更新变量guesses和hits。下面首先来检查用户输入是否有效；如果有效，就将变量guesses加1。然后编写代码检查是否击中了战舰。

```
// 变量声明


while (isSunk == false) {
    guess = prompt("Ready, aim, fire! (enter a number from 0-6):");
    if (guess < 0 || guess > 6) {
       alert("Please enter a valid cell number!");
    } else {
       guesses = guesses + 1;
    }
}
```

核实猜测的位置是否是0～6，以检查用户输入的有效性。

如果输入无效，就使用alert向用户指出这一点。

如果输入有效，就将guesses加1，以记录用户猜测了多少次。

下面详细地介绍一下有效性测试。你知道要检查输入是否为0～6，但这个条件到底是如何测试这一点的呢？下面来详细解释。

这个条件的含义如下：如果用户的猜测小于0或大于6，这个条件就为true，此时输入是无效的。

```
if (guess < 0 || guess > 6) {
```

这个条件实际上由两个测试组成，其中第一个测试检查guess是否小于0。

第二个测试检查guess是否大于6。

这是OR运算符，将两个测试合而为一，使得只要有一个测试为true，整个条件就为true。如果两个测试都为false，则整个条件为false；此时guess为0～6，意味着它是有效的。

世上没有愚蠢的问题

问：我注意到函数prompt显示的对话框有一个取消按钮。如果用户单击该按钮，函数prompt将返回什么呢？

答：如果用户单击这个对话框中的取消按钮，函数prompt将返回null（而不是一个字符串）。别忘了，null表示没有值；就这里而言，这是合适的，因为用户取消了而没有输入值。我们可利用这一点，通过检查返回值是否为null来判断用户是否单击了取消按钮；如果单击了该按钮，就可以结束游戏。在这里的代码中，我们没有这样做，但请务必牢记这一点，因为本书后面可能这样做。

问：你说过，函数prompt总是返回一个字符串。既然如此，我们怎么能将字符串（如"0"或"6"）同数字（如0或6）进行比较呢？

答：在这种情况下，为了执行比较guess<0和guess>6，JavaScript将尝试把guess存储的字符串转换为数字。只要用户输入的是数字（如4），JavaScript就知道在必要时如何将字符串（如"4"）转换为数字。本书后面将更详细地介绍类型转换这一主题。

问：如果用户在prompt对话框中输入的不是数字（如"six"或"quit"），结果将如何呢？

答：在这种情况下，JavaScript无法将字符串转换为数字，进而将"six"或"quit"同6进行比较。这种比较的结果为false，导致计算机认为未击中战舰。在更健壮的战舰游戏版本中，我们将更详尽地检查用户输入，确保用户输入的是数字。

问：使用OR运算符时，有一个条件为true或两个条件都为true时，整个条件就为true吗？

答：是的，这两种情况都为true。使用OR运算符（||）时，如果有一个条件为true或两个条件都为true，则结果为true；如果两个条件都为false，则结果为false。

问：有AND运算符吗？

答：有！AND运算符（&&）的工作原理与OR运算符类似，但仅当两个测试都为true时，结果才为true。

问：何为无限循环？

答：问得好。无限循环是困扰程序员的众多问题之一。别忘了，循环必须包含条件测试，而只要循环测试为true，循环就继续。如果代码没有修改任何可导致条件测试最终变为false的因素，循环将不断进行下去，直到终止浏览器或重启计算机。

布尔运算符两分钟学习指南

布尔运算符用于结果为true或false的布尔表达式中。有两种布尔运算符：比较运算符和逻辑运算符。

比较运算符

比较运算符对两个值进行比较。下面是一些常见的比较运算符：

- `<` 表示小于
- `>` 表示大于
- `==` 表示等于
- `===` 表示正好等于（后面将详细介绍）
- `<=` 表示小于或等于
- `>=` 表示大于或等于
- `!=` 表示不等于

逻辑运算符

逻辑运算符将两个布尔表达式合而为一，得到一个布尔结果（true或false）。下面是两个逻辑运算符：

- `||` 表示OR（或）。只要至少有一个表达式为true，结果就为true。
- `&&` 表示AND（与）。仅当两个表达式都为true时，结果才为true。

另一个逻辑运算符是NOT，它作用于一个布尔表达式（而不是两个）：

- `!` 表示NOT（非）。仅当表达式为false时，结果才为true。

判断是否击中

- [x] 创建循环并获取用户的输入
- [ ] 检查用户的猜测
- [ ] 检查战舰是否被击沉
- [ ] 向用户显示统计信息

这比较有趣：用户猜测战舰的位置，而我们需要判断他是否猜对了。具体地说，我们需要判断猜测的位置是否与战舰所处的单元格相同；如果相同，就将变量hits加1。

下面是击中检测代码的第一个版本，我们来详细解释它。

```
if (guess == location1) {
     hits = hits + 1;
} else  if (guess == location2) {
     hits = hits + 1;
} else if (guess == location3) {
     hits = hits + 1;
}
```

如果guess与location1相等，说明击中了战舰，因此将变量hits加1。

如果guess与location2相等，做同样的处理。

最后，如果guess与location3相等，也需要将变量hits加1。

如果上述三个条件都不满足，就不修改变量hits。

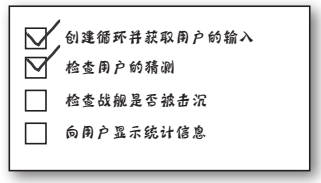
注意到我们在if/else代码块中使用了缩进。这让代码更容易理解，在有大量嵌套代码块时尤其如此。



磨笔上阵

你觉得这个版本的击中检测代码怎么样？是不是看起来比需要的复杂？我们重复代码的方式是否显得有些冗余？能够对其进行简化吗？你能够使用||运算符（布尔OR运算符）的知识简化这些代码吗？继续往下阅读前，务必查看本章末尾的答案。

添加击中检测代码



下面来将前几页的代码整合在一起。

```
// 变量声明

while (isSunk == false) {
    guess = prompt("Ready, aim, fire! (enter a number from 0-6):");
    if (guess < 0 || guess > 6) {
      alert("Please enter a valid cell number!");
    } else {
      guesses = guesses + 1;
    }
    if (guess == location1 || guess == location2 || guess == location3) {
        hits = hits + 1;
    }
  }
}
```

循环：只要战舰未被击沉

获取用户的猜测

检查用户的猜测。

将猜测次数**加**1

用户的猜测是有效的，因此将猜测次数加1。

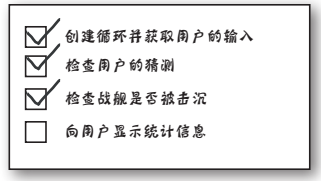
如果用户猜测的是战舰位置之一

将击中次数**加**1

如果用户猜测的是战舰位置之一，就将击中计数器加1。

我们使用||（OR）运算符将三个条件合而为一，并将其用于一条if语句中。这条语句的含义如下：如果guess等于location1、location2或location3，就将hits加1。

你击沉了我的战舰



就要大功告成了，整个游戏的逻辑都差不多完成了。再看一眼伪代码，现在需要做的是检查战舰是否被击中了三次。如果是，战舰就被击沉了，而如果战舰被击沉，就需要将`isSunk`设置为`true`，并将这一点告诉用户。下面来编写这样的代码，再将其整合到前面的代码中。

首先检查是否击中了三次。

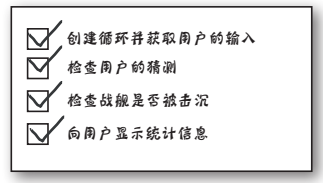
再看一眼前面的循环。如果isSunk为true，结果将如何呢？

```
if (hits == 3) {
    isSunk = true;
    alert("You sank my battleship!");
}
```

如果击中了三次，就将isSunk设置为true。

同时让用户知道这一点！

进行游戏后分析



isSunk设置为true后，while循环就会结束。确实是这样，我们熟悉的程序就要停止执行while循环体了，不知不觉间游戏就要结束了。我们还需向用户显示一些有关其表现的统计信息，代码如下。

```
var stats = "You took " + guesses + " guesses to sink the battleship, " +
            "which means your shooting accuracy was " + (3/guesses);
alert(stats);
```

我们创建一个字符串，其中包含要向用户显示的消息，这包括用户猜测了多少次和射击精度。注意，为插入变量*guesses*并分散到多行中，这里使用拼接运算符（+）将多个部分合并成一个字符串。你现在只需按原样输入即可，稍后我们将更详细地解释这些代码。

下面将这些代码和击沉检测代码与其他代码整合起来。

```
// 变量声明

while (isSunk == false) {
    guess = prompt("Ready, aim, fire! (enter a number from 0-6):");
    if (guess < 0 || guess > 6) {
        alert("Please enter a valid cell number!");
    } else {
        guesses = guesses + 1;

        if (guess == location1 || guess == location2 || guess == location3) {
            hits = hits + 1;

            if (hits == 3) {

                isSunk = true;

                alert("You sank my battleship!");
            }
        }
    }
}
var stats = "You took " + guesses + " guesses to sink the battleship, " +
            "which means your shooting accuracy was " + (3/guesses);
alert(stats);
```

循环：只要战舰未被击沉

获取用户的猜测

将猜测次数**加**1

如果用户猜测的是战舰位置之一

将击中次数**加**1

如果击中了3次

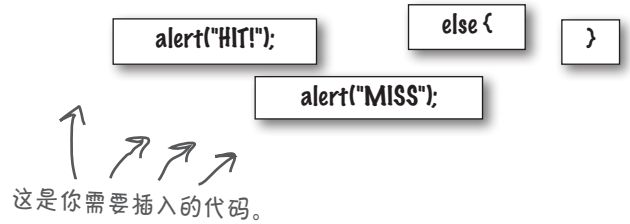
将isSunk**设置**为true

向用户**显示** You sank my battleship!

向用户**显示**统计信息



前面说过，伪代码并非总是完美无缺的。实际上，伪代码遗漏了一些内容：没有指出用户的猜测对不对。你能将指出这一点的代码插入正确的地方吗？

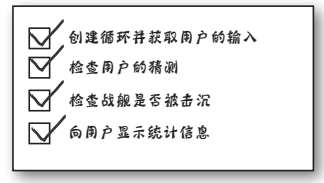



```
// 变量声明

while (isSunk == false) {
    guess = prompt("Ready, aim, fire! (enter a number from 0-6):");
    if (guess < 0 || guess > 6) {
        alert("Please enter a valid cell number!");
    } else {
        guesses = guesses + 1;
        if (guess == location1 || guess == location2 || guess == location3) {
            hits = hits + 1;
            if (hits == 3) {
              isSunk = true;
              alert("You sank my battleship!");
            }
        }
    }
}
var stats = "You took " + guesses + " guesses to sink the battleship, " +
               "which means your shooting accuracy was " + (3/guesses);
alert(stats);
```

这里有很多右花括号。如果你无法确定它们各自对应于哪个左花括号，就在书上划线来确定这一点。

完整的游戏逻辑



好了，我们现在将伪代码都转换为实际的JavaScript代码了，还发现并补全了伪代码遗漏的内容。下面是完整的JavaScript代码，请输入它们并存储到文件battleship.js中。

```
var location1 = 3;
var location2 = 4;
var location3 = 5;
var guess;
var hits = 0;
var guesses = 0;
var isSunk = false;

while (isSunk == false) {
    guess = prompt("Ready, aim, fire! (enter a number from 0-6):");
    if (guess < 0 || guess > 6) {
        alert("Please enter a valid cell number!");
    } else {
        guesses = guesses + 1;

        if (guess == location1 || guess == location2 || guess == location3) {
            alert("HIT!");
            hits = hits + 1;
            if (hits == 3) {
                isSunk = true;
                alert("You sank my battleship!");
            }
        } else {
           alert("MISS");
        }
    }
}
var stats = "You took " + guesses + " guesses to sink the battleship, " +
               "which means your shooting accuracy was " + (3/guesses);
alert(stats);
```

一点点质量保证

质量保证（quality assurance，QA）是指对软件进行测试以找出其中的缺陷。下面来对这些代码做简单的测试。准备就绪后，在浏览器中加载battleship.html，并开始游戏。尝试各种不同的操作。程序能完美地运行还是出现了问题？如果出现了问题，请在右边列出它们。下面是我们的测试情况。

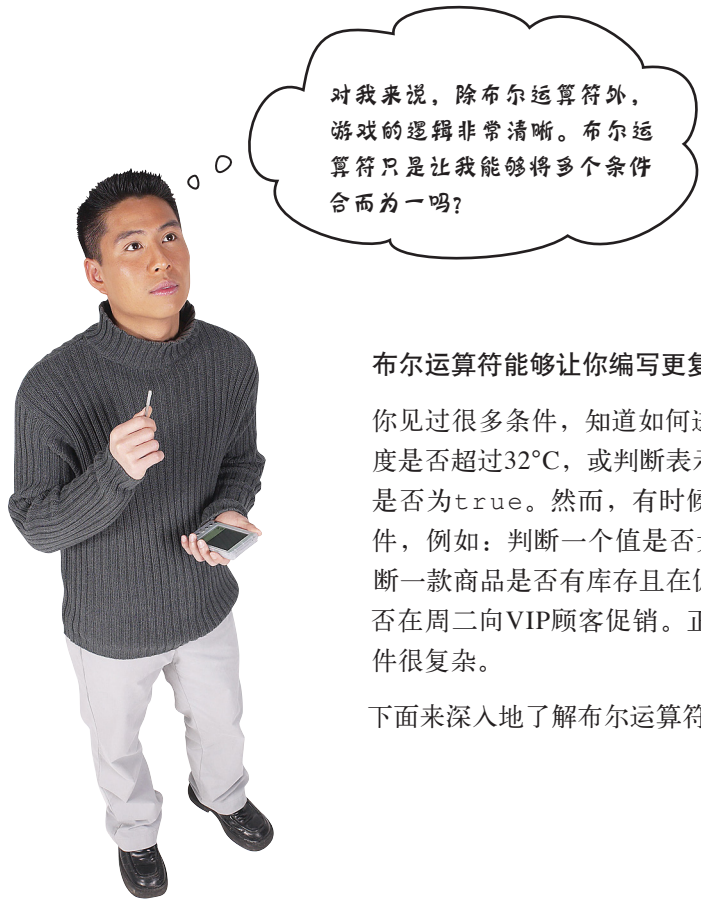


如果遇到不符合预期的情况或者发现可改进的地方，都将它们粗略地记录下来。

下面是我们与游戏交互的情况。



图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权



布尔运算符能够让你编写更复杂的逻辑语句。

你见过很多条件，知道如何进行测试了，如判断温度是否超过32°C，或判断表示商品库存情况的变量是否为`true`。然而，有时候需要测试更复杂的条件，例如：判断一个值是否大于32且小于100；判断一款商品是否有库存且在促销；判断一款商品是否在周二向VIP顾客促销。正如你看到的，这些条件很复杂。

下面来深入地了解布尔运算符的工作原理。

假设要判断一款商品是否有库存且在促销，可这样编写代码。

```
if (inStock == true) {
    if (onSale == true) {
        // 好像很划算！
        alert("buy buy buy!");
    }
}
```

首先，检查这款商品是否有库存。

如果有库存，再检查它是否在促销。

如果在促销，就采取相应的措施，如买一些！

请注意，仅当两个条件都满足时，才会执行这行代码！

可将这两个条件合而为一，以简化上述代码。在简单战舰游戏中，我们检查`guess<0`或`guess>6`；而这里我们要判断的是：`inStock`为`true`且`onSale`为`true`。下面来看看如何将这两个条件合而为一。

这是AND运算符。使用AND时，仅当两部分都为true时，整个条件才为true。

```
if (inStock == true  &&  onSale == true) {
        // 好像很划算!
        alert("buy buy buy!");
}
```

这里的代码不仅更简洁，还更容易理解。只要将这些代码与前一页的代码进行比较，你就能看出来。

我们还可以更进一步，以多种方式组合使用不同的布尔运算符：

这里在一个条件表达式中同时使用了AND和OR运算符。这个条件表达式的含义如下：如果一款商品有库存，且这款商品在促销或价格低于60，就买下它。

```
if (inStock == true  &&  (onSale == true || price < 60)) {
        // 好像很划算!
        alert("buy buy buy!");
}
```

注意到这里使用括号对条件进行了编组，旨在先获得OR运算的结果，再根据它计算AND运算的结果。



磨笔上阵

下面是一系列需要计算结果的布尔表达式。请在空白处填写你的计算结果，并查看本章末尾的答案，再继续往下阅读。

```
var temp = 81;
var willRain = true;
var humid = (temp > 80 && willRain == true);
```

humid的值是什么？____

```
var guess = 6;
var isValid = (guess >= 0 && guess <= 6);
```

isValid的值是什么？____

```
var kB = 1287;
var tooBig = (kB > 1000);
var urgent = true;
var sendFile =
    (urgent == true || tooBig == false);
```

sendFile的值是什么？____

```
var keyPressed = "N";
var points = 142;
var level;
if (keyPressed == "Y" ||
    (points > 100 && points < 200)) {
   level = 2;
} else {
   level = 1;
}
```

level的值是什么？____

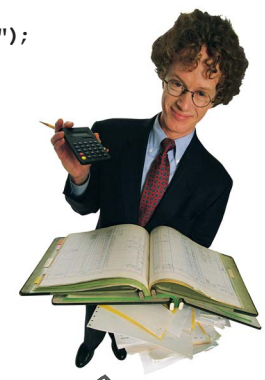


练习



Bob和Bill都是会计，他们要为公司网站开发一个价格检查应用程序。他们都使用布尔表达式编写了`if/else`语句，而且信心满满，认为自己编写的代码正确无误。哪个会计编写的代码是正确的呢？这两个会计适合从事编码工作吗？请给出你的答案，再查看本章末尾的答案，然后继续往下阅读。

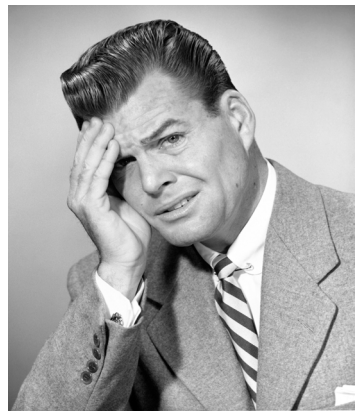
```
if (price < 200 || price > 600) {
   alert("Price is too low or too high! Don't buy the gadget.");
} else {
   alert("Price is right! Buy the gadget.");
}
```




```
if (price >= 200 || price <= 600) {
   alert("Price is right! Buy the gadget.");
} else {
   alert("Price is too low or too high! Don't buy the gadget.");
}
```


图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

别这么啰嗦好不好



我真不知道怎么说好，你在指定条件时一直都比较啰嗦。这是什么意思呢？请看下面的条件：

指定条件时，我们通常将布尔变量与true或false进行比较。

```
if    (inStock  ==  true)  {
    ...
}
```

可inStock就是一个值为true或false的布尔变量。

实际上，这样做有点多余。指定条件时，根本目标是确保其值为true或false，可布尔变量inStock就是这样的值。因此，不需要将这个变量与任何值进行比较，可以直接使用它。换句话说，可将前面的代码修改成下面这样：

```
if    (inStock)  {
    ...
}
```

仅使用布尔变量本身时，只要该变量为true，条件测试就为true，进而执行相应的代码块。

如果inStock为false，条件测试将失败，进而跳过相应的代码块。

虽然有人认为比较啰嗦的原始版本更清晰地表达了其意图，但在实际代码中更常用的是简洁版。另外，简洁版也更容易理解。



练习

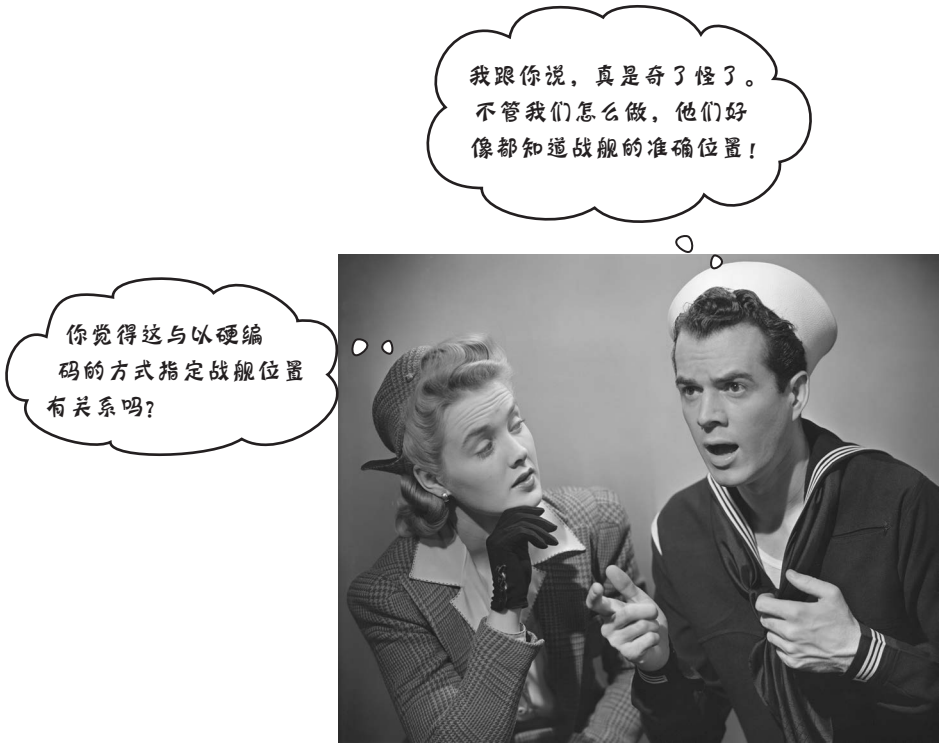
下面两条语句都根据变量onSale和inStock来计算变量buyIt的值。请根据变量inStock和onSale的各种可能取值，确定这两条语句计算得到的buyIt值。哪条语句导致buyIt的值为true的可能性更大呢？

```
var buyIt = (inStock || onSale);
```

onSale	inStock	buyIt	buyIt
true	true		
true	false		
false	true		
false	false		



```
var buyIt = (inStock && onSale);
```



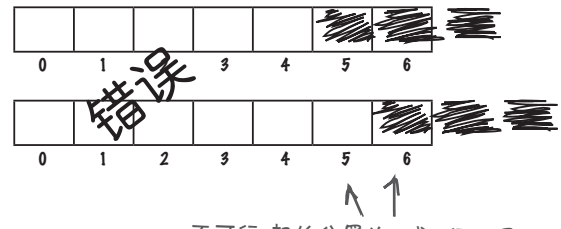
完善简单的战舰游戏

还有一个小问题需要处理，因为当前你以硬编码的方式指定战舰的位置：每次玩游戏时，战舰都位于单元格3～5处。对测试而言，这没有任何问题，但为让游戏更有趣，必须随机地指定战舰的位置。

下面来想想，在包含7个单元格的一维网格中，如何正确地放置战舰呢？我们需要一个起始位置，它让我们能够将战舰放置到相连的三个单元格中，这意味着起始位置必须是0～4。



起始位置可以是0、1、2、3或4，这样余下的空间才能容纳占据三个单元格的战舰。



不可行 起始位置为5或6行不通

如何随机地指定位置

确定起始位置（0～4）后，就可使用它和接下来的两个单元格来放置战舰。

使用随机位置和接下来的两个单元格。

```
var location1 = randomLoc;
var location2 = location1 + 1;
var location3 = location2 + 1;
```

那么如何生成随机数呢？这需要求助于JavaScript内置函数。具体地说，JavaScript有很多内置的数学函数，其中有两个可用于生成随机数。内置函数和一般意义的函数将在本书后面更详细地介绍。就现在而言，我们将直接使用这些函数来完成任务。

举世闻名的随机数生产配方

我们先来看函数Math.random。通过调用这个函数，可获得一个随机的小数：

变量randomLoc。我们想要将一个0～4的数字赋给这个变量。

Math.random包含在标准JavaScript中，它返回一个随机数。

```
var randomLoc = Math.random();
```

唯一的问题是，它返回的是0～1（不包括1）的小数，如0.128、0.830、0.9、0.42，因此我们需要想办法将其转换为0～4的随机数。



我们需要的是0～4的整数（即0、1、2、3或4），而不是小数（如0.34）。为此，可将Math.random返回的数字乘以5，以获得更接近目标的数字。下面的代码说明了这一点：

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

如果将随机数乘以5，将得到一个0～5（不包括5）的数字，如0.139 83、4.231、2.3451或4.999。

```
var randomLoc = Math.random() * 5;
```

别忘了，*表示乘法运算。

这更接近目标了！现在需要做的是，将小数部分删除，得到一个整数。为此，可使用另一个数学函数——Math.floor：

我们可以使用Math.floor将数字向下圆整为最接近的整数。

```
var randomLoc = Math.floor(Math.random() * 5);
```

例如，这将把0.139 83转换为0，把2.34转换为2，把4.999转换为4。

世上没有愚蠢的问题

问：我们要生成一个0～4的数字，为何在代码中乘以5（如`Math.floor(Math.random() * 5)`）呢？

答：问得好。首先，Math.random生成0～1（不包括1）的数字。因此使用Math.random可生成的最大数字为0.999…，乘以5后，可得到的最大数字为4.999…。Math.floor总是向下圆整，即将1.2圆整为1，将1.999也圆整为1。如果我们生成0～5（不包括5）的数字，圆整后将为0～4。这并非唯一的做法，其他语言中的做法通常也不同，但在JavaScript代码中，这是最常见的做法。

问：这么说，如果要生成0～100（包括100）的随机数，就可使用代码`Math.floor(Math.random() * 101)`了？

答：完全正确！通过乘以101，再使用Math.floor向下圆整，可确保结果最大为100。

问：`Math.random()`中的括号是做什么用的？

答：调用函数时都需要使用括号。有时候需要给函数传递值，如使用alert显示消息时；而有时候不需要这样做，如使用函数Math.random时。然而，只要调用函数（无论是内置函数还是非内置函数），都需要使用括号。现在不要操心这一点，下一章将介绍这些细节。

问：我编写的战舰游戏不能正确地运行：在网页中，除标题Play battleship外，什么都看不到。如何找出我在什么地方做错了呢？

答：这是控制台的用武之地。如果你犯了错，比如忘记用引号将字符串括起，JavaScript通常会指出程序的语法不正确，还可能指出问题出在哪一行。然而，有些错误更难发现。例如，如果你将isSunk = false错写成了isSunk == false，JavaScript将不会显示任何错误消息，但代码不会像期望的那样运行。对于这种错误，可尝试在代码的各个地方使用console.log来显示变量的值，看看能不能找出错误。

再来一点点质量保证



需要的全部代码都编写好了。下面使用它们来替换原来的位置指定代码，如下所示。然后进行几次测试运行，看看你能以多快的速度击沉战舰。

```
var randomLoc = Math.floor(Math.random() * 5);
var location1 = randomLoc;
var location2 = location1 + 1;
var location3 = location2 + 1;
var guess;
var hits = 0;
var guesses = 0;
var isSunk = false;

while (isSunk == false) {
    guess = prompt("Ready, aim, fire! (enter a number from 0-6):");
    if (guess < 0 || guess > 6) {
      // 其他代码……
```

将原来的位置变量声明和赋值语句替换为这些新语句。

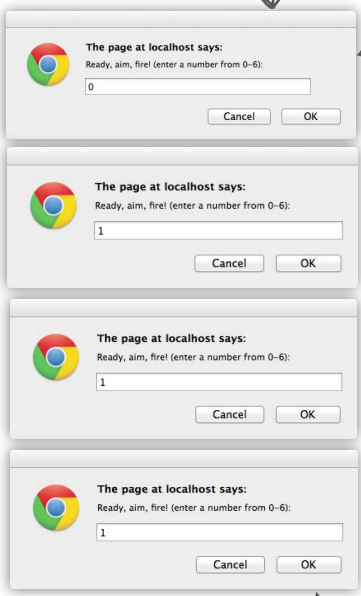
下面是我们进行的一次测试。随机指定战舰的位置后，游戏更有趣了，但我们依然获得了相当不错的得分。



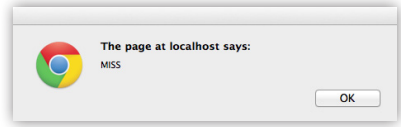


等等，我们注意到好像有什么地方不对。提示：我们依次输入0、1、1和1时，结果不太对劲！你能找出其中的原因吗？

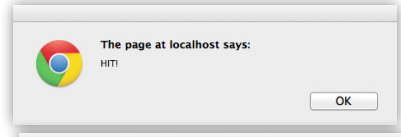
下面是我们的猜测过程。



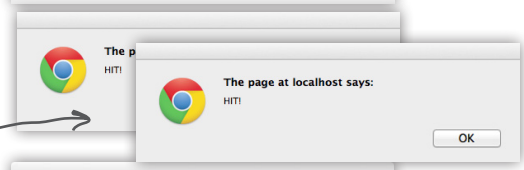
第一次没有猜对。

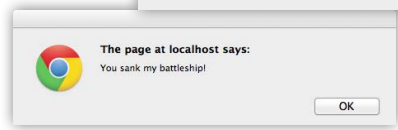


第二次猜测时，我们找到了战舰的位置之一。

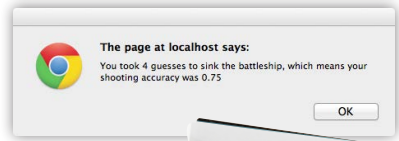


然后我们不断地输入这个位置，每次都击中了战舰！

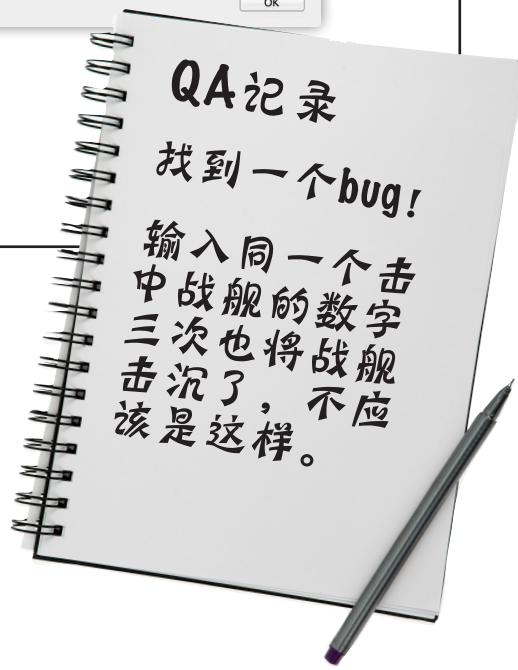




击中第三次后，战舰被击沉了！但好像有什么地方不对：击中同一个地方三次应该不能将战舰击沉。



我们输入0、1、1和1，而战舰位于单元格1～3处。





扣人心弦

我们能**找到**这个bug吗？

我们能**修复**这个bug吗？

本书后面将提供完善得多的战舰游戏版本，敬请期待！

现在看看你能想出办法将这个bug修复吗？

说说代码重用

祝贺你编写了第一个真正的JavaScript程序。你可能注意到了，我们使用了几个**内置函数**，如alert、prompt、console.log和Math.random。这些函数让你能够弹出对话框、将输出写入控制台以及生成随机数，而你需要做的工作很少。这就像变魔术一样。这些内置函数是为你编写好的现成代码，你可根据需要反复使用，而使用时只需调用即可。



对于函数，需要学习的东西很多：如何调用它们，可给它们传递什么类型的值，等等。下一章将着手全面介绍这些主题，你也将学习如何创建自己的函数。

但进入下一章之前，还有一些要点需要复习，一个填字游戏需要完成。另外，别忘了睡个好觉，将本章的内容都消化消化。



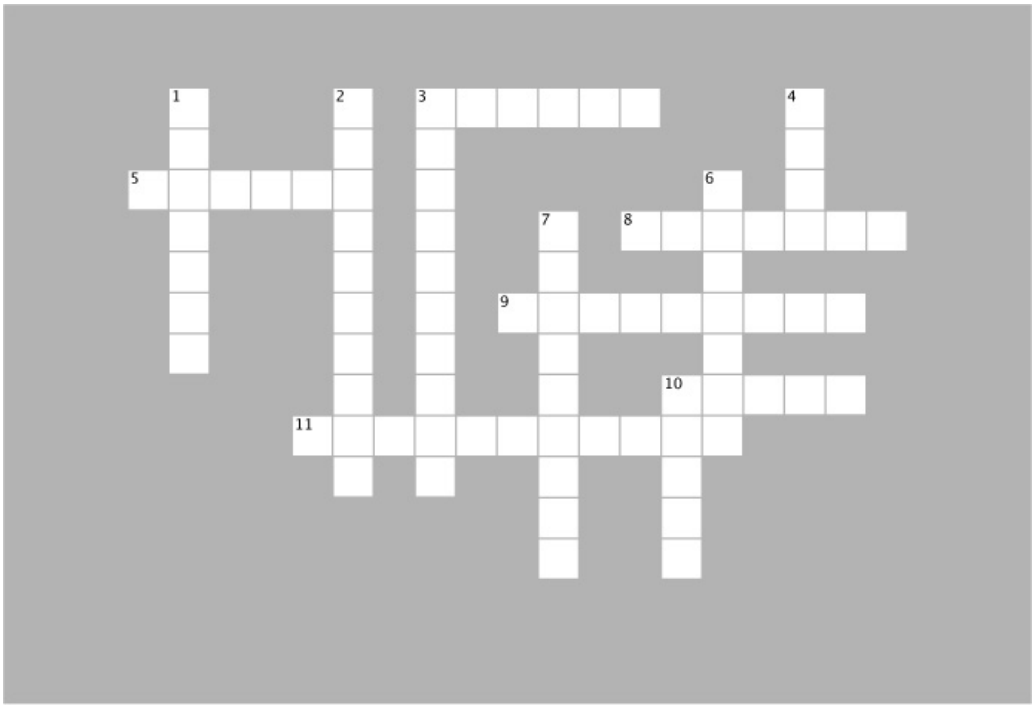
要点

- 可使用流程图来指出决策点以及要采取的措施，从而大致描述JavaScript程序的逻辑。
- 编写程序前，最好使用伪代码粗略地描绘一下程序需要完成的任务。
- **伪代码**粗略地描绘了实际代码需要完成的任务。
- 有两类布尔运算符：比较运算符和逻辑运算符。用于表达式时，布尔运算符的结果为true或false。
- **比较运算符**对两个值进行比较，结果为true或false。例如，可像下面这样使用比较运算符<（小于运算符）：3 < 6。这个表达式的结果为true。
- **逻辑运算符**合并两个布尔值。例如，true || false的结果为true，而true && false的结果为false。
- 可使用函数Math.random来生成0～1（包括0，但不包括1）的随机数。
- 函数Math.floor将小数向下圆整为最接近的整数。
- 使用Math.random和Math.floor时，Math中的M务必大写，不要小写。
- JavaScript函数prompt显示一个对话框，其中包含一条消息以及供用户输入值的空间。
- 在本章中，我们使用了prompt来获取用户输入，并使用了alert在浏览器中显示战舰游戏的结果。



JavaScript填字游戏

填字游戏对学习JavaScript有何帮助呢？头脑体操会将JavaScript知识深深地烙印在你的脑海中！



横向

3. 用于获取用户输入的函数。

5. 用于随机地指定战舰位置的函数（句点之后部分）。

8. 记录战舰是否被击沉的变量的类型。

9. 未初始化的变量的值。

10. 在布尔运算符的两种可能结果中，除`true`外的另一种结果。

11. `while`和`if`语句都使用的测试。

纵向

1. 如果你擅长测试程序，可能想成为什么保证方面的专家？

2. 用于检查两个值是否相等的==所属的布尔运算符类型。

3. 一种帮助你理解程序将如何运行的代码。

4. 要让AND运算符（`&&`）的结果为`true`，条件的两部分都必须是什么值？

6. 运算符OR（`||`）和AND（`&&`）所属的布尔运算符类型。

7. `alert`和`prompt`等都属于JavaScript的内置什么（复数形式）？

10. 要让OR运算符（`||`）的结果为`false`，条件的两部分都必须是什么值？



假设我们的虚拟网格类似于下面这样：



并像下面这样使用了位置变量来指出战舰的位置：

```
location1 = 3;
location2 = 4;
location3 = 5;
```

而用户的输入序列如下：

```
1, 4, 2, 3, 5
```

在给定上述用户输入的情况下，前一页的伪代码将如何执行呢？请将你认为的结果写在下面。我们指出了开始时的情况。答案如下：

location1	location2	location 3	guess	guesses	hits	isSunk
3	4	5	——	0	0	false
3	4	5	1	1	0	false
3	4	5	4	2	1	false
3	4	5	2	3	1	false
3	4	5	3	4	2	false
3	4	5	5	5	3	true



下面两条语句都根据变量onSale和inStock来计算变量buyIt的值。请根据变量inStock和onSale的各种可能取值，确定这两条语句计算得到的buyIt值。哪条语句导致buyIt的值为true的可能性更大呢？使用OR运算符（||）的那条！



```
var buyIt = (inStock || onSale);
```

onSale	inStock	buyIt	buyIt
true	true	true	true
true	false	true	false
false	true	true	false
false	false	false	false

```
var buyIt = (inStock && onSale);
```



下面是一系列需要计算结果的布尔表达式，请在空白处填写你的计算结果。答案如下：

```
var temp = 81;
var willRain = true;
var humid = (temp > 80 && willRain == true);
```

humid的值是什么？ true

```
var guess = 6;
var isValid = (guess >= 0 && guess <= 6);
```

isValid的值是什么？ true

```
var kB = 1287;
var tooBig = (kB > 1000);
var urgent = true;
var sendFile =
    (urgent == true || tooBig == false);
```

sendFile的值是什么？ true

```
var keyPressed = "N";
var points = 142;
var level;
if (keyPressed == "Y" ||
    (points > 100 && points < 200)) {
   level = 2;
} else {
   level = 1;
}
```

level的值是什么？ 2



Bob和Bill都是会计，他们要为公司网站开发一个价格检查应用程序。他们都使用布尔表达式编写了`if/else`语句，而且都信心满满，认为自己编写的代码正确无误。哪个会计编写的代码是正确的呢？这两个会计适合从事编码工作吗？答案如下。




```
if (price < 200 || price > 600) {
    alert("Price is too low or too high! Don't buy the gadget.");
} else {
    alert("Price is right! Buy the gadget.");
}
```




```
if (price >= 200 || price <= 600) {
    alert("Price is right! Buy the gadget.");
} else {
    alert("Price is too low or too high! Don't buy the gadget.");
}
```

Bob的编码水平更高（可能会计工作也做得更好）。Bob的解决方案可行，但Bill的解决方案行不通。为什么呢？我们使用三个不同的价格（太低、太高和刚刚好）来检验Bob和Bill编写的条件，看看得到的结果是什么。

价格	Bob	Bill
100	true alert: Don't buy!	true alert: Buy!
700	true alert: Don't buy!	true alert: Buy!
400	false alert: Buy!	true alert: Buy!

如果价格为100，则Bob编写的条件为true（因为100小于200；同时别忘了，使用OR运算符时，只要一个表达式为true，整个表达式就为true），因此使用alert显示不购买。

然而，Bill编写的条件也为true，因为价格小于或等于600，因此整个表达式为true，进而让用户购买，虽然价格太低了。

上述结果表明，不管价格如何，Bill编写的条件总是为true。因此他编写的代码总是让用户购买！Bill还是安心地做会计吧。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权



前面说过，伪代码并非总是完美无缺的。实际上，伪代码遗漏了一些内容：没有指出用户的猜测对不对。你能将指出这一点的代码插入正确的地方吗？答案如下。

```
// 变量声明

while (isSunk == false) {
    guess = prompt("Ready, aim, fire! (enter a number from 0-6):");
    if (guess < 0 || guess > 6) {
        alert("Please enter a valid cell number!");
    } else {
        guesses = guesses + 1;
        if (guess == location1 || guess == location2 || guess == location3)
{
            alert("HIT!");
            hits = hits + 1;
            if (hits == 3) {
              isSunk = true;
              alert("You sank my battleship!");
            }
        } else {
            alert("MISS");
        }
    }
}
var stats = "You took " + guesses + " guesses to sink the battleship, " +
               "which means your shooting accuracy was " + (3/guesses);
alert(stats);
```



磨笔上阵答案

你觉得这个版本的击中检测代码怎么样？是不是看起来比需要的复杂？我们重复代码的方式是否显得有些冗余？能够对其进行简化吗？你能够使用||运算符（布尔OR运算符）的知识简化这些代码吗？答案如下。

```
if (guess == location1) {
     hits = hits + 1;
} else  if (guess == location2) {
     hits = hits + 1;
} else if (guess == location3) {
     hits = hits + 1;
}
```

我们反复使用了相同的代码。

如果要调整hits的修改方式，就必须修改代码的三个地方。这常常是导致代码出现bug和问题的根源。

不仅如此，这段代码还比需要的复杂。它更难理解，需要输入的代码更多。

通过使用OR运算符，可将所有的测试合而为一，使得在猜测的位置为loaction1、loaction2或loaction3时，if条件就为true，进而更新变量hits。

```
if (guess == location1 || guess == location2 || guess == location3) {
     hits = hits + 1;
}
```

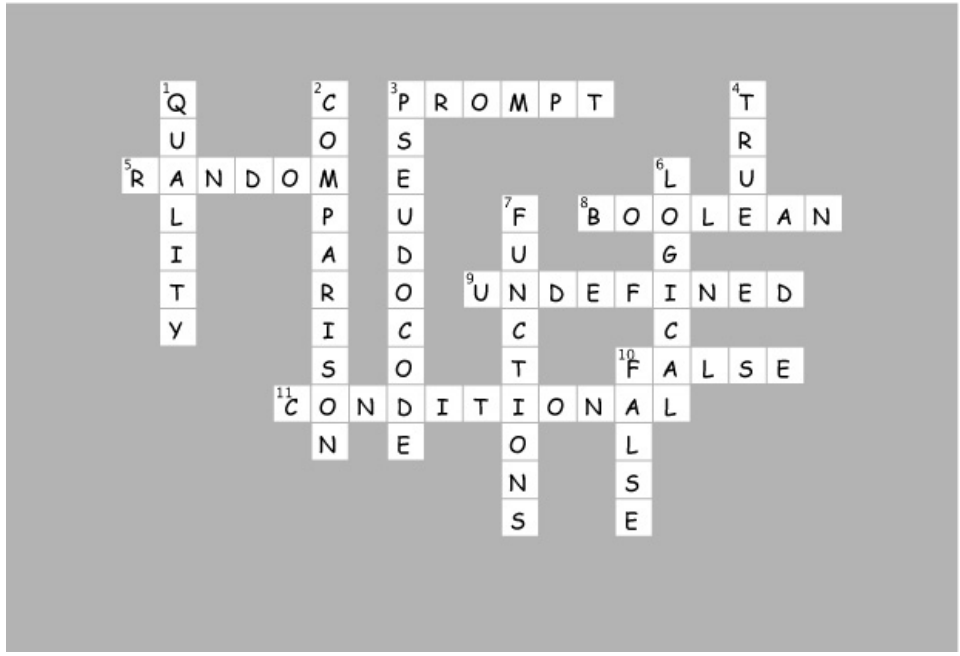
除更容易理解外，是不是看起来也舒服得多？

另外，如果需要调整hits的修改方式，只需要在一个地方这样做，更不容易出错。



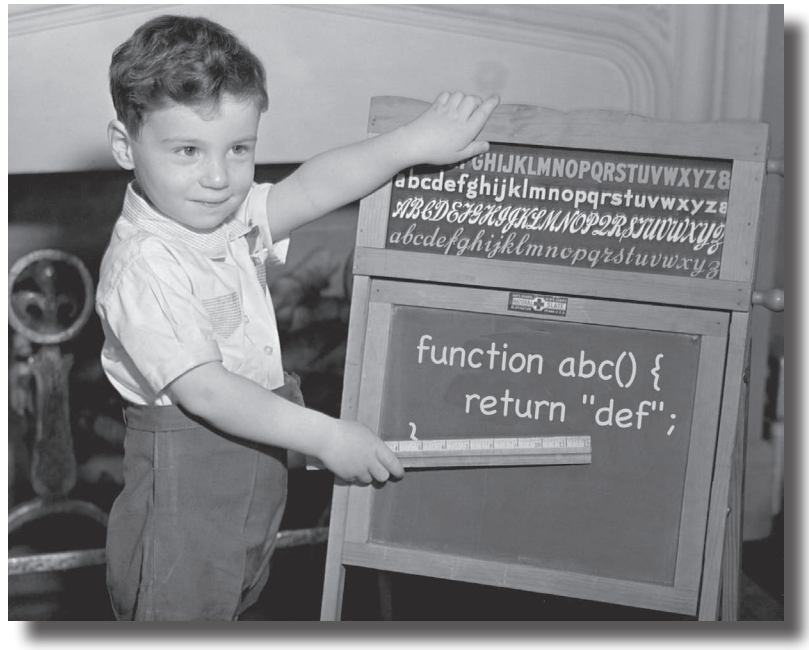
JavaScript填字游戏答案

填字游戏对学习JavaScript有何帮助呢？头脑体操会将这些知识深深烙印在你的脑海中！答案如下。



3 函数简介

养成函数思维



为使用你的第一个超级武器作好准备。你编写过一些代码，现在该使用函数来提高效率了。通过使用**函数**，你可编写适用于各种不同环境的代码。这些代码可反复重用且管理起来容易得多。你还可以将通用代码抽取出来，给它指定一个简单的名称，这样就能将复杂的东西抛诸脑后，将精力放在重要的内容上。你将发现，函数不仅是脚本编写人员通往程序员的大门，还是JavaScript编程风格的核心。本章介绍函数的基本知识：机制以及工作原理的方方面面。在本书余下的篇幅中，你将不断提高函数方面的技能。下面就来为此打下坚实的基础吧。

本书后面将更详细地介绍这个方面。



磨笔上阵

请简单地分析下面的代码。它给你什么样的感觉？请选择后面你认为合适的评价，你也可以写下自己的评价。

```
var dogName = "rover";
var dogWeight = 23;
if (dogWeight > 20) {
   console.log(dogName + " says WOOF WOOF");
} else {
   console.log(dogName + " says woof woof");
}
dogName = "spot";
dogWeight = 13;
if (dogWeight > 20) {
   console.log(dogName + " says WOOF WOOF");
} else {
   console.log(dogName + " says woof woof");
}
dogName = "spike";
dogWeight = 53;
if (dogWeight > 20) {
   console.log(dogName + " says WOOF WOOF");
} else {
   console.log(dogName + " says woof woof");
}
dogName = "lady";
dogWeight = 17;
if (dogWeight > 20) {
   console.log(dogName + " says WOOF WOOF");
} else {
   console.log(dogName + " says woof woof");
}
```

- ☐ A. 重复的代码多得不得了。
- ☐ B. 如果要修改输出的显示方式或添加其他小狗属性，将需要做大量重复的工作。
- ☐ C. 输入起来很繁琐！
- ☐ D. 可读性不佳。
- ☐ E. _____

这些代码到底有什么问题呢

正如你看到的，有些代码被**反复使用**。这有什么问题吗？从表面上看，什么问题都没有，毕竟它们能够正确地运行，不是吗？下面来深入研究这些代码。

```
var dogName = "rover";
var dogWeight = 23;
if (dogWeight > 20) {
    console.log(dogName + " says WOOF WOOF");
} else {
    console.log(dogName + " says woof woof");
}
  ...
dogName = "lady";
dogWeight = 17;
if (dogWeight > 20) {
    console.log(dogName + " says WOOF WOOF");
} else {
    console.log(dogName + " says woof woof");
}
```

将小狗的体重与20进行比较，如果超过了20，就发出较大的叫声；否则就发出较小的叫声。

这些代码与前面完全相同。在后面，这些代码还反复出现了很多次。

```
dogName = "spike";
dogWeight = 53;
if (dogWeight > 20) {
   console.log(dogName + " says WOOF WOOF");
} else {
   console.log(dogName + " says woof woof");
}
dogName = "lady";
dogWeight = 17;
if (dogWeight > 20) {
   console.log(dogName + " says WOOF WOOF");
} else {
   console.log(dogName + " says woof woof");
}
```

诚然，这些代码看起来一点问题都没有，但编写起来很繁琐，阅读起来很痛苦，而且会在需要修改时带来麻烦。随着你的编程经验日益丰富，最后一个问题将更加明显：随着时间的推移，所有的代码都需要修改，而上面的代码是你迟早要面对的噩梦，因为你在其中反复编写了同样的逻辑。如果需要修改这种逻辑，就必须修改多个地方。另外，程序越大，需要修改的地方就会越多，因此犯错的可能性也就越大。我们应该做的是，想办法将这样的代码提取出来，放在一个地方，以便需要时可轻松地重用（reuse）它们。



如何改进这些代码呢？花几分钟想出几种可能的解决方案。JavaScript提供了可助你一臂之力的特性吗？



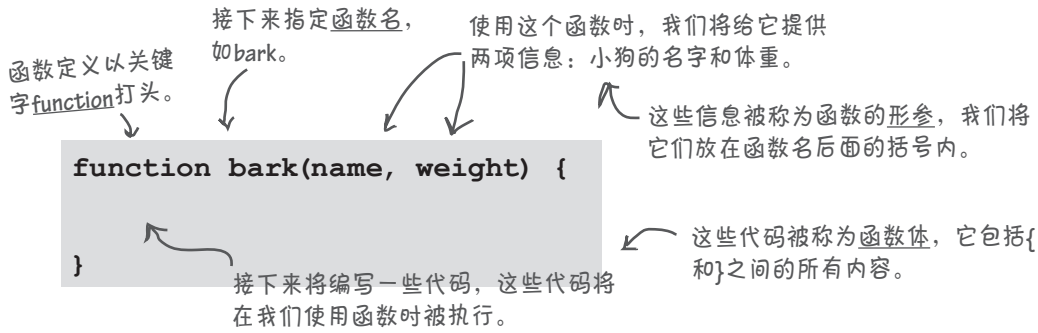
如果能够找到一种代码重用办法，使得需要时能够直接使用（而不用重复输入），可以给代码指定好记的名称以方便记忆，并在修改时只需修改一个地方（而不是很多地方），那就太好了。但我知道这不过是白日做梦。

以前提到过函数吗

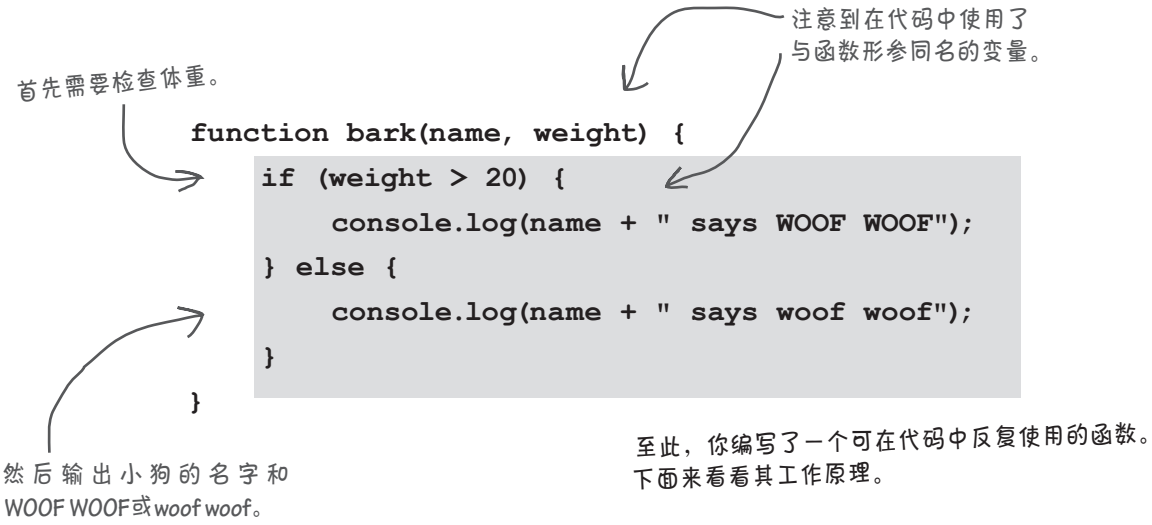


来认识一下**函数**。JavaScript函数让你能够给一系列代码指定名称，以便需要时反复使用它们。这好像正是我们需要的药方。

假设你要编写一些反复“发出叫声”的代码：在小狗较大时发出较大的叫声（显示大写的WOOF WOOF），而小狗较小时发出较小的叫声（显示小写的woof woof）。之后需要在代码中多次使用这种发出叫声的功能。下面来编写一个可反复使用的bark函数。



现在需要给这个函数编写代码，用来检查小狗的体重，并输出相应的叫声。



图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

函数到底是如何工作的呢

先来重新编写前面的代码，在其中使用新定义的函数bark。

```
function bark(name, weight) {
   if (weight > 20) {
      console.log(name + " says WOOF WOOF");
   } else {
      console.log(name + " says woof woof");
   }
}

bark("rover", 23);
bark("spot", 13);
bark("spike", 53);
bark("lady", 17);
```

这很好，所有的代码逻辑都放在一个地方。

现在只需调用函数bark几次，并将小狗的名字和体重传递给它即可。

代码简单得多了！

哇，代码少多了，而且对需要修改这些代码的同事来说，也容易理解多了。另外，所有逻辑都放在了一个方便的地方。

这很好，但这些代码是如何协同工作的呢？下面来详细解读。

首先，我们定义了函数。

我们在代码开头定义了函数bark。浏览器读取这些代码时，发现这是一个函数，进而查看函数体中的语句。浏览器知道，不应马上执行这个函数的语句，而要等到在代码的其他地方调用了这个函数时再执行它们。

另外，注意到函数是**参数化**的，这意味着它被调用时将接受小狗的名字和体重。这让你能够针对任意数量的小狗调用这个函数。每次调用时，都将对传入的名字和体重应用指定的逻辑。

这些是形参，将在调用函数时给它们赋值。

```
function bark(name, weight) {
   if (weight > 20) {
      console.log(name + " says WOOF WOOF");
   } else {
      console.log(name + " says woof woof");
   }
}
```

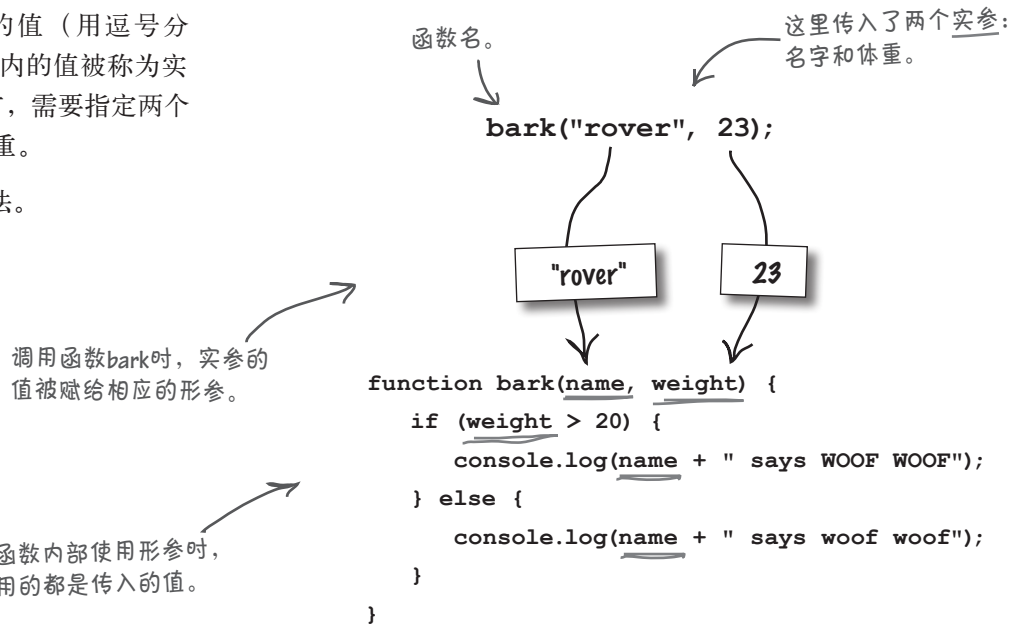
函数内的所有代码都属于函数体。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

下面来调用这个函数。

要调用函数，只需依次指定函数名、左括号、需要传入的值（用逗号分隔）和右括号。括号内的值被称为实参，就函数bark而言，需要指定两个实参：狗的名字和体重。

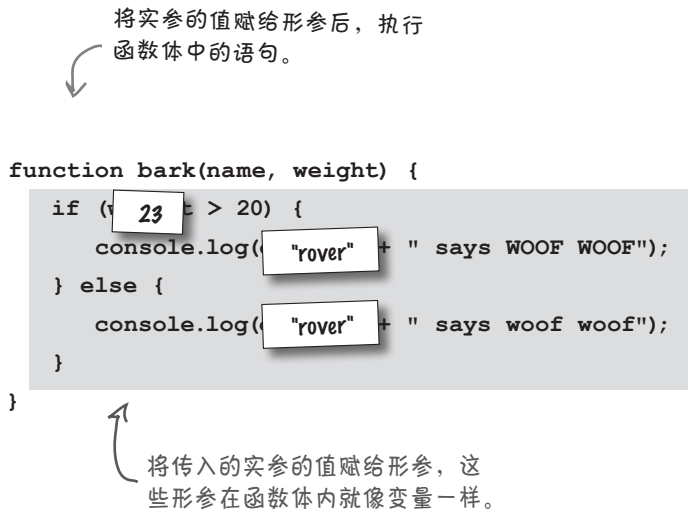
下面是调用的具体方法。



调用函数后，函数体将完成所有的工作。

知道每个形参的值（如name的值为"rover"，而weight的值为23）后，就可以执行函数体了。

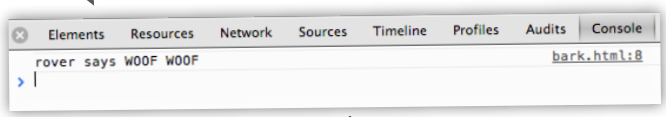
就像我们在前面编写的所有代码一样，函数体中的语句也是按从上到下的顺序执行的。唯一的差别是，将传给函数的实参的值赋给了形参name和weight。



图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

函数执行完毕后……执行函数体的逻辑后（在这个示例中，Rover的体重为23磅，它发出较大的叫声——显示大写的WOOF WOOF），函数就执行完毕了。函数执行完毕后，将返回到调用函数bark的语句之后继续执行。

使用浏览器的开发工具来访问控制台，以便能够看到函数bark的输出。



"rover says WOOF WOOF"

```
function bark(name, weight) {
   if (weight > 20) {
      console.log(name + " says WOOF WOOF");
   } else {
      console.log(name + " says woof woof");
   }
}
bark("rover", 23);
bark("spot", 13);
bark("spike", 53);
bark("lady", 17);
```

我们只是这样做，

再这样做。

函数执行完毕后，浏览器将接着执行在调用该函数的语句之后的代码。

这里使用不同的实参再次调用了函数bark，因此将从头开始执行这个函数！

磨笔上阵

下面是一些调用函数bark的代码。在每个调用旁边指出该调用是否会导致错误；如果不会，就写出其输出。继续往下阅读前，请查看本章末尾的答案。

写出你认为将在控制台中显示的内容。

```
bark("juno", 20);     _____
bark("scottie", -1);  _____
bark("dino", 0, 0);   _____
bark("fido", "20");   _____
bark("lady", 10);     _____
bark("bruno", 21);    _____
```

你知道这些调用的结果如何吗？



代码冰箱贴

冰箱上贴着一些被打乱的JavaScript代码。你能将这些代码重组为一个生成如下输出的程序吗？请注意，冰箱上的有些代码可能是多余的，也就是说你可能不会用到所有的冰箱贴。答案见本章末尾。

```
,
,
,
}
}
}
(
function
)
{
whatShallIWear(80);
else {
    console.log("Wear t-shirt");
}
whatShallIWear
else if (temp < 70) {
    console.log("Wear a sweater");
}
temperature
if (temp < 60) {
    console.log("Wear a jacket");
}
temp
whatShallIWear(60);
whatShallIWear(50);
```

JavaScript控制台

```
Wear a jacket
Wear a t-shirt
Wear a sweater
```

我们以这种方式表示通用控制台。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权



起底函数

本周访谈：函数不为人知的一面

Head First：欢迎函数！我们将深入挖掘，给你来个全面曝光。

函数：很高兴接受访谈。

Head First：我们注意到，很多JavaScript新手对你不闻不问。他们直接动手编写代码，从上到下一行又一行，根本就不使用函数。你真的不可或缺吗？

函数：我可厉害了，这些新手错失了良机，真是太遗憾了。你可以这样看我：让你只需编写代码一次，就可反复重用它们。

Head First：恕我不恭，如果你只是让他们能够反复做同样的事情，是不是有点枯燥？

函数：不，函数是参数化的。换句话说，你每次使用函数时都传入实参，从而执行相关的计算。

Head First：比方说呢？

函数：假设你要向用户指出其购物车中商品的总价，可编写函数`computeShoppingCartTotal`。然后，你就可以将很多用户的购物车传递给这个函数，而它每次都将计算相应购物车中商品的总价。

Head First：既然你这么厉害，为何很多新手不用你？

函数：其实不是这样的，他们一直都在使用我：`alert`、`prompt`、`Math.random`、`document.write`。如果不使用函数，很难编写出有用的代码。不使用函数的新手并不多，这些新手只是不定义**自己的**函数而已。

Head First：`alert`和`prompt`是函数，这很好理解，但`Math.random`看起来不太像函数。

函数：`Math.random`也是函数，只是刚好关联到了新手用得不多的另一样东西——**对象**。

Head First：哦，是的，是对象。读者在本书后面将学习它们。

函数：那太好了，届时我就不用再费口舌了。

Head First：咱们来说说实参和形参吧，它们有点不太好理解。

函数：可以这么理解：在整个函数体中，每个形参都像变量一样。调用函数时，传入的每个值都将赋给相应的形参。

Head First：实参呢？

函数：它指的就是传递给函数的值，是函数调用的参数。

Head First：你看起来好像也没那么厉害。我是说你让我能够重用代码，还让我能够给形参传递值。如果仅此而已，我不觉得你有什么神秘的。

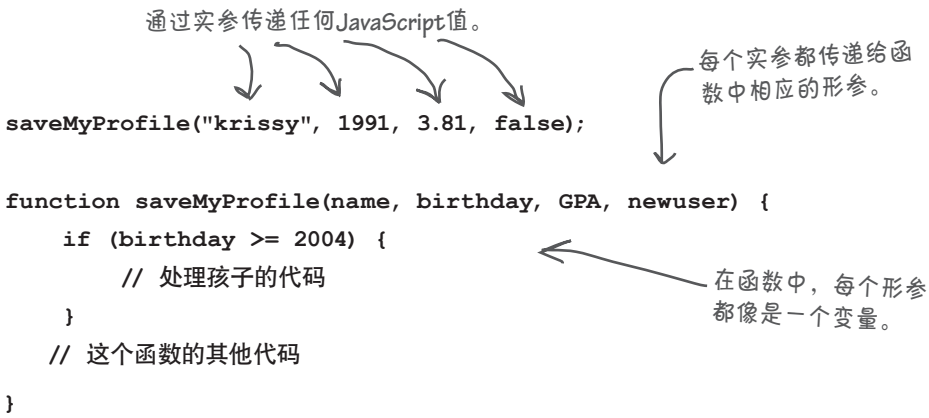
函数：这只是基本功能，还有很多其他的功能：我能够返回值，我能够将你的代码伪装成匿名的，我能够实现闭包，我还与对象关系紧密。

Head First：真的吗？！咱们能否再来一次访谈，专门谈谈这种关系？

函数：没问题。

可以向函数传递哪些东西

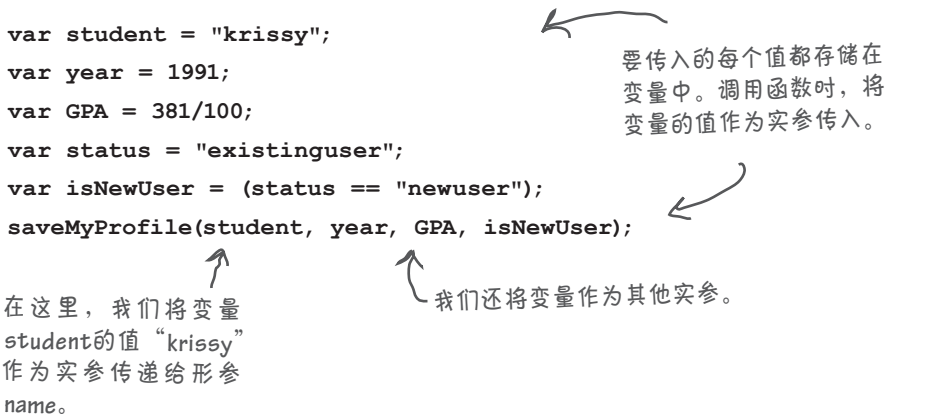
你调用函数时向它传递实参，这些实参对应于函数定义中的形参。
可通过实参传递任何JavaScript值，如字符串、布尔值或数字。




```
saveMyProfile("krissy", 1991, 3.81, false);

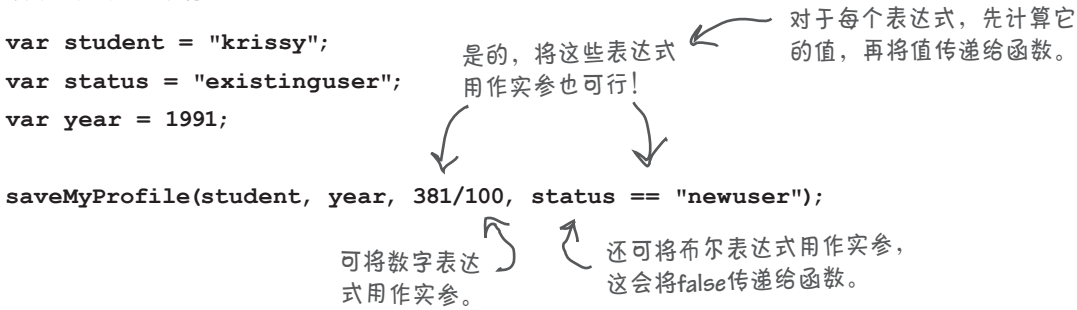
function saveMyProfile(name, birthday, GPA, newuser) {
    if (birthday >= 2004) {
        // 处理孩子的代码
    }
   // 这个函数的其他代码
}
```

还可以将变量作为实参传递，这种情况其实更常见。下面的函数调用与前面的函数调用相同，但将变量用作实参。



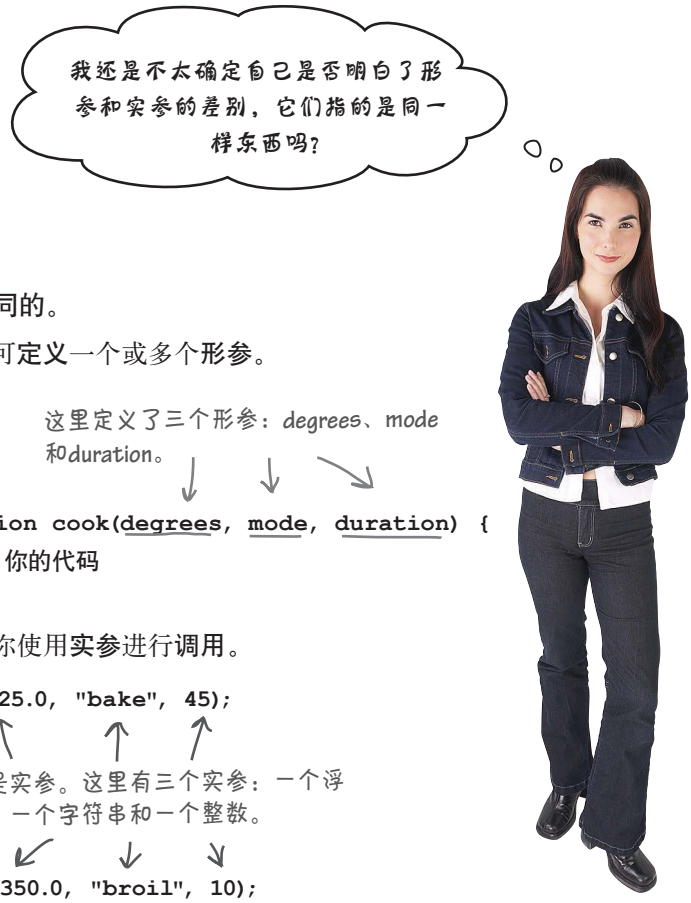

```
var student = "krissy";
var year = 1991;
var GPA = 381/100;
var status = "existinguser";
var isNewUser = (status == "newuser");
saveMyProfile(student, year, GPA, isNewUser);
```

还可以将表达式用作实参。




```
var student = "krissy";
var status = "existinguser";
var year = 1991;

saveMyProfile(student, year, 381/100, status == "newuser");
```



不，它们是不同的。

定义函数时，可**定义**一个或多个**形参**。

这里定义了三个形参：degrees、mode和duration。

```
function cook(degrees, mode, duration) {
    // 你的代码
}
```

调用函数时，你使用**实参**进行**调用**。

```
cook(425.0, "bake", 45);
```

这些是实参。这里有三个实参：一个浮点数、一个字符串和一个整数。

```
cook(350.0, "broil", 10);
```

因此，形参只需定义一次，但你可能会多次调用函数，而且每次提供的实参可能不同。

考考你的脑力

下述代码的输出是什么？你确定吗？

```
function doIt(param) {
    param = 2;
}
var test = 1;
doIt(test);
console.log(test);
```



下面是一些JavaScript代码，其中包含变量、函数定义和函数调用。你的任务是找出所有的变量、函数、实参和形参，并将它们填在右边相应的方框中。继续往下阅读前，请查看本章末尾的答案。

```
function dogYears(dogName, age) {
    var years = age * 7;
    console.log(dogName + " is " + years + " years old");
}
var myDog = "Fido";
dogYears(myDog, 4);

function makeTea(cups, tea) {
    console.log("Brewing " + cups + " cups of " + tea);
}
var guests = 3;
makeTea(guests, "Earl Grey");

function secret() {
    console.log("The secret of life is 42");
}
secret();

function speak(kind) {
    var defaultSound = "";
    if (kind == "dog") {
        alert("Woof");
    } else if (kind == "cat") {
        alert("Meow");
    } else {
        alert(defaultSound);
    }
}
var pet = prompt("Enter a type of pet: ");
speak(pet);
```

变量

函数

内置函数

实参

形参

JavaScript按值传递实参

这意味着传递的是实参的副本

明白JavaScript如何传递实参很重要。JavaScript**按值传递实参**（pass-by-value），这意味着把每个实参的值**复制**给形参。要了解这是如何传递的，来看一个简单的示例。

1 声明变量age，并将其初始化为7。

```
var age = 7;
```



2 声明函数addOne，它包含参数x，并将x的值加1。

```
function addOne(x) {
    x = x + 1;
}
```



3 下面来调用函数addOne，并将变量age作为实参传递给它。age的值被复制给形参x。

```
addOne(age);
```

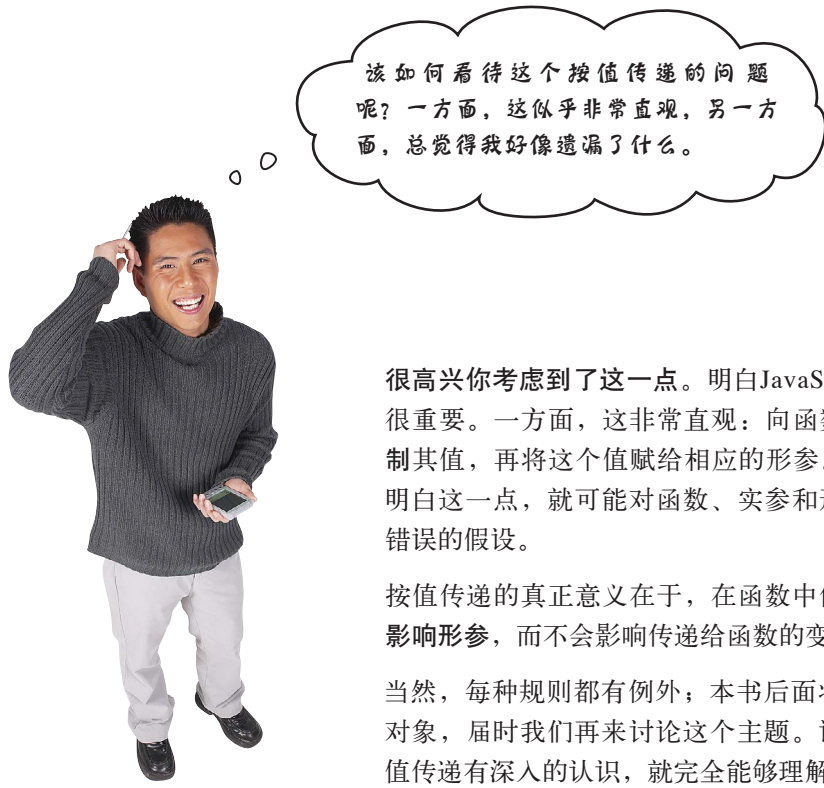


4 现在x的值被加上了1。但别忘了，x是age的副本，因此只有x的值被加1，而age的值不变。

```
function addOne(x) {
    x = x + 1;
}
```

将x的值加1。





很高兴你考虑到了这一点。明白JavaScript如何给函数传递值很重要。一方面，这非常直观：向函数传递实参时，首先**复制**其值，再将这个值赋给相应的形参。另一方面，如果你不明白这一点，就可能对函数、实参和形参如何协同工作做出错误的假设。

按值传递的真正意义在于，在函数中修改形参的值时，**只会影响形参**，而不会影响传递给函数的变量。差不多就是这样。

当然，每种规则都有例外；本书后面将通过两章的篇幅介绍对象，届时我们再来讨论这个主题。请不要担心，只要对按值传递有深入的认识，就完全能够理解后面的讨论。

就目前而言，只需牢记由于实参是按值传递的，**在函数中处理形参时，其影响范围将限制在函数中**。这有点像拉斯维加斯[①]。

①上一句黑体部分原文为：“Whatever happens to a parameter in the function, stays in the function.”套用拉斯维加斯的城市宣传语“What happens in Vegas stays in Vegas.”。——编者注

重温 考考你的脑力

还记得这个“考考你的脑力”吗？理解按值传递后，你现在的想法是不是不同了。你第一次猜对了吗？

```
function doIt(param) {
    param = 2;
}
var test = 1;
doIt(test);
console.log(test);
```


图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

错误地调用函数

前面的函数调用都正确无误，但如果传递给函数的实参太多或不够，结果将如何呢？听起来好像很危险，咱们来看看。



实验1：如果传入的实参不够，结果将如何？

这样做好像很危险，但实际情况只是将没有相应实参的形参设置为未定义，如下所示：

```
function makeTea(cups, tea) {
  console.log("Brewing " + cups + " cups of " + tea);
}
makeTea(3);
```

JavaScript控制台

```
Brewing 3 cups of undefined
```

注意到形参tea的值为undefined，这是因为我们没有给它传递值。

实验2：如果传递的实参太多，结果将如何？

在这种情况下，JavaScript将忽略多余的实参，如下所示：

```
function makeTea(cups, tea) {
  console.log("Brewing " + cups + " cups of " + tea);
}
makeTea(3, "Earl Grey", "hey ma!", 42);
```

JavaScript控制台

```
Brewing 3 cups of Earl Grey
```

一点问题都没有。函数忽略多余的实参。

实际上，有办法确定传递的实参是否太多，但现在我们暂时不去管它。

实验3：没有形参时结果将如何？

不用担心，很多函数都没有形参！

```
function barkAtTheMoon() {
  console.log("Wooooooooooooo!");
}
barkAtTheMoon();
```

JavaScript控制台

```
Wooooooooooooo!
```

函数还可返回值

你知道如何与函数通信，即知道如何向函数传递实参，但另一个方向呢？函数能够与你通信吗？咱们来看看return语句。

这是函数bake，它将烤箱温度（单位为度）作为参数。

```
function bake(degrees) {
    var message;

    if (degrees > 500) {
        message = "I'm not a nuclear reactor!";
    } else if (degrees < 100) {
        message = "I'm not a refrigerator!";
    } else {
        message = "That's a very comfortable temperature for me.";
        setMode("bake");
        setTemp(degrees);
    }
    return message;
}

var status = bake(350);
```

它根据形参degrees存储的温度值将一个变量设置为相应的字符串。

假设实际工作是由这些函数完成的，但这里暂时不考虑它们的细节。

我们关心的是这条return语句，它将message作为函数的结果返回。

这个函数被调用并返回时，作为结果返回的字符串将被赋给变量status。

在这个示例中，如果打印变量status，将发现它存储的是字符串“That's a very comfortable temperature for me.”。请仔细研究这些代码，确认你明白为什么会这样！

我烤出美味饼干的最佳温度是350°C。请尝试使用其他的温度调用函数bake，再接着往下阅读。



图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

详解包含return语句的函数的执行过程

至此，你知道了实参和形参的工作原理，还知道如何从函数返回值。下面来跟踪一个函数的完整执行过程，看看一路上的情况。请务必按顺序查看下面的步骤。

① 首先，我们声明了变量radius并将其初始化为5.2。

② 接下来，调用函数calculateArea，并将变量radius作为实参传递给它。

③ 这个实参的值被传递给形参r，而函数calculateArea开始执行，此时r的值为5.2。

④ 执行函数体。首先，声明了变量area，然后检查形参r的值是否小于或等于0。

⑤ 如果r <= 0，就从函数返回0，而函数也将停止执行。然而，我们传入的是5.2，因此不会执行这行代码。

⑥ 相反，将执行else子句。

⑦ 使用形参r的值5.2计算圆的面积。

⑧ 从函数返回面积值。这将停止执行函数，并返回指定的值。

⑨ 将函数返回的值存储在变量theArea中。

⑩ 接着执行下一行代码。

```
③ function calculateArea(r) {
      var area;
④     if (r <= 0) {
⑤       return 0;
⑥     } else {
⑦       area = Math.PI * r * r;
⑧       return area;
      }
  }
① var radius = 5.2;
②
⑨ var theArea = calculateArea(radius);

⑩ console.log("The area is: " + theArea);
```

输出如下！

JavaScript控制台

```
The area is: 84.94866535306801
```

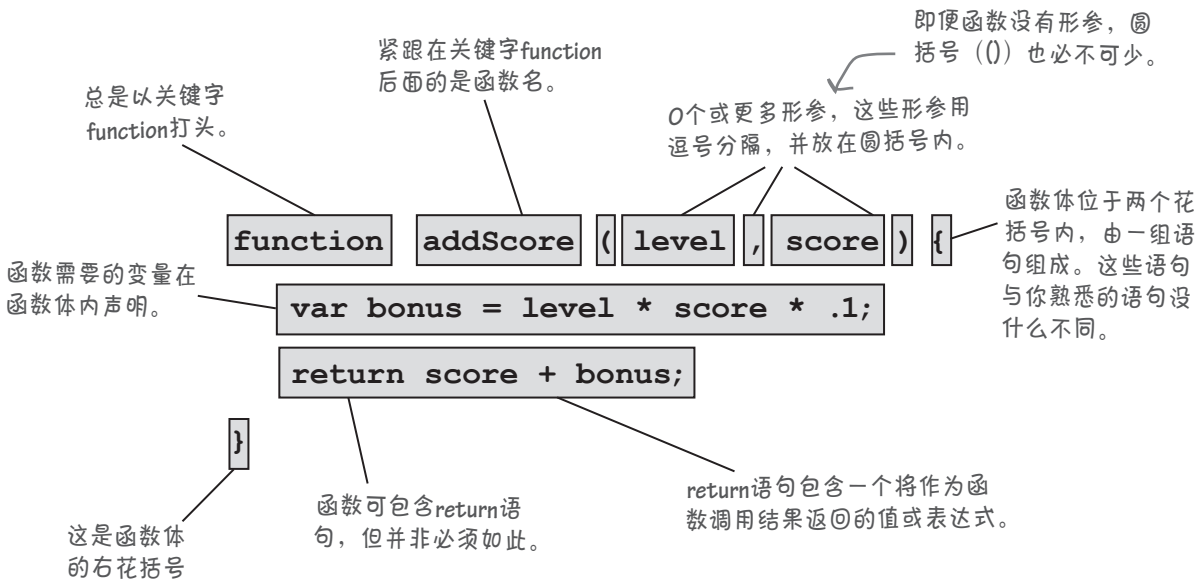
开发人员通常称之为“跟踪执行过程”或简称为“跟踪”。正如你看到的，调用函数和返回值时，执行流程可能会有跳跃。请慢慢地跟踪，一步一步来。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权



函数剖析

知道如何定义和调用函数后，咱们来透彻地解释相关的语法。下面全面地剖析了函数的各个组成部分：



世上没有愚蠢的问题

问：如果实参的顺序不对，将错误的实参传递给了形参，结果将如何？

答：很难说，但几乎可以肯定，这要么导致代码不正确，要么导致运行阶段错误。务必仔细查看函数的形参，以确定函数要求传入哪些实参。

问：形参名前为何没有关键字var？形参不是一个新变量吗？

答：形参实际上就是一个新变量，但这种变量的实例化工作都是由函数替你完成的，因此你不需要在形参名前指定关键字var。

问：有哪些函数命名规则？

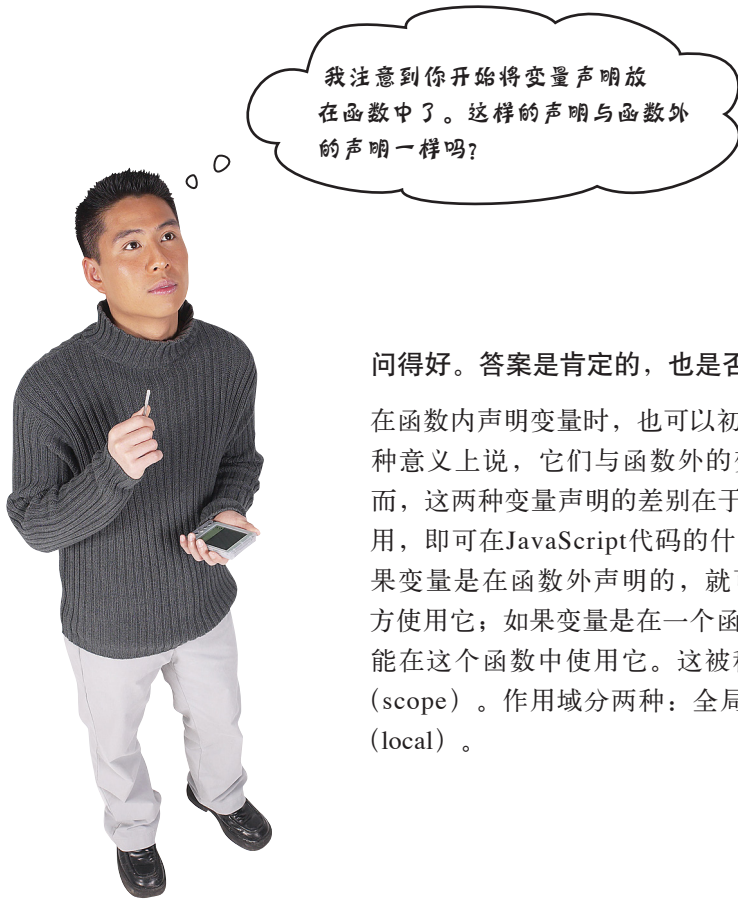
答：函数命名规则与变量命名规则相同。与变量一样，应使用对你来说有意义的函数名，并指出函数的作用。你还可以在函数名中使用骆驼式拼写法（如camelCase）来组合多个单词。

问：如果用作实参的变量与形参同名（如将它们都命名为x），结果将如何？

答：即便实参和形参同名，如都为x，实参x也将为形参x的副本。因此它们是两个不同的变量，修改形参x的值不会导致实参x的值发生变化。

问：如果函数没有return语句，它将返回什么？

答：没有return语句的函数返回undefined。



问得好。答案是肯定的，也是否定的。

在函数内声明变量时，也可以初始化新变量，从这种意义上说，它们与函数外的变量声明相同。然而，这两种变量声明的差别在于变量在什么地方可用，即可在JavaScript代码的什么地方引用它。如果变量是在函数外声明的，就可在代码的任何地方使用它；如果变量是在一个函数中声明的，就只能在这个函数中使用它。这被称为变量的**作用域**（scope）。作用域分两种：全局（global）和局部（local）。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权



全局变量和局部变量

一定要明白它们的差别，不然会露怯哟

你知道，可在JavaScript代码的任何地方使用关键字var和名称来声明变量：

```
var avatar;
var levelThreshold = 1000;
```

这些是全局变量，在JavaScript代码的任何地方都可访问它们。

你看到了，还可在函数中声明变量：

```
function getScore(points) {
    var score;
    var i = 0;
    while (i < levelThreshold) {
        // 你的代码

        i = i + 1;
    }
    return score;
}
```

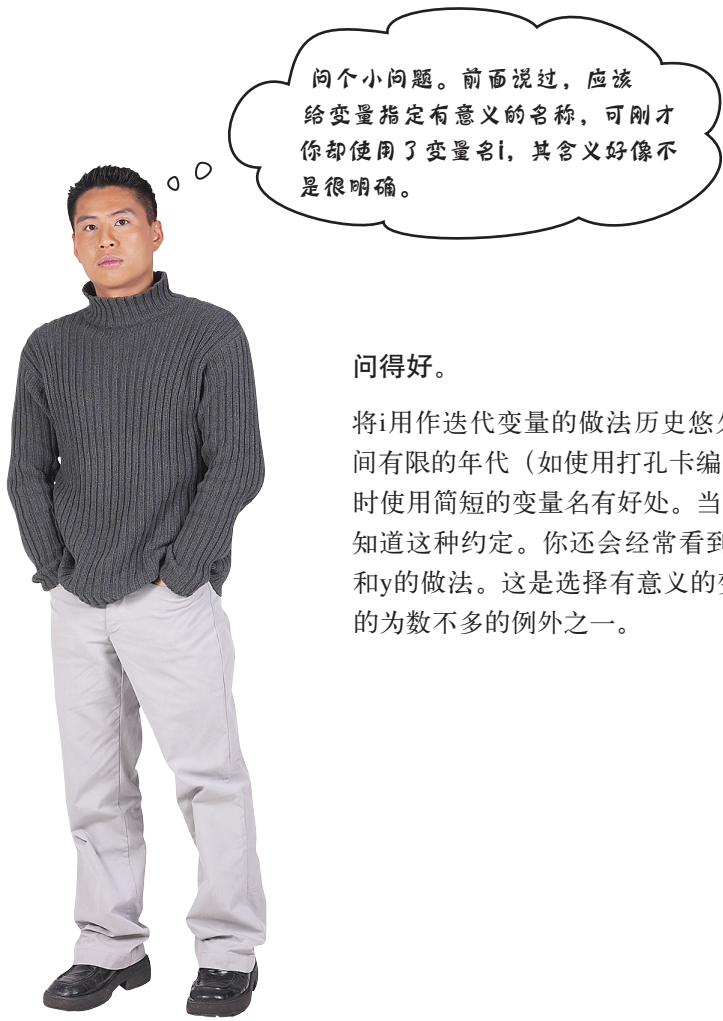
变量points、score和i都是在函数中声明的。

这些变量被称为局部变量，因为它们只在当前函数中可用。

虽然我们在函数中使用了变量levelThreshold，但它也是全局的，因为它是在函数外面声明的。

如果变量是在函数外声明的，它就是全局变量；如果变量是在函数中声明的，它就是局部变量。

这有什么关系吗？不都是变量吗？变量的声明位置决定了它在代码的哪些地方是可见的。熟悉这两种变量的工作原理不仅有助于你理解别人编写的代码，还有助于你编写出更易于维护的代码。



问得好。

将i用作迭代变量的做法历史悠久。这种约定始于空间有限的年代（如使用打孔卡编写代码的年代），那时使用简短的变量名有好处。当前，所有的程序员都知道这种约定。你还会经常看到这样使用j、k乃至x和y的做法。这是选择有意义的变量名这种最佳实践的为数不多的例外之一。

了解局部变量和全局变量的作用域

变量的定义位置决定了其**作用域**，即变量在代码的哪些地方可见，哪些地方不可见。咱们来看一个例子，其中有作用域为全局的变量，也有作用域为局部的变量。（别忘了，在函数外定义的变量的作用域为全局，而在函数中定义的变量的作用域为局部。）

```
var avatar = "generic";
var skill = 1.0;
var pointsPerLevel = 1000;
var userPoints = 2008;

function getAvatar(points) {
    var level = points / pointsPerLevel;

    if (level == 0) {
       return "Teddy bear";
    } else if (level == 1) {
       return "Cat";
    } else if (level >= 2) {
       return "Gorilla";
    }
}

function updatePoints(bonus, newPoints) {
    var i = 0;
    while (i < bonus) {
        newPoints = newPoints + skill * bonus;
        i = i + 1;
    }
    return newPoints + userPoints;
}

userPoints = updatePoints(2, 100);
avatar = getAvatar(2112);
```

这4个变量的作用域是局部的。这意味着它们在后续所有代码中都是已定义且可见的。

请注意，如果你在网页中链接到了其他的脚本，它们将能够看到这些全局变量，而你也能看到它们的全局变量！

这里的变量level是局部变量，只在函数getAvatar中可见。这意味着只有这个函数能够访问变量level。

别忘了形参points，它也是一个局部变量，其作用域为函数getAvatar。

注意到函数getAvatar还使用了全局变量pointsPerLevel。

在函数updatePoints中，我们定义了局部变量i，它对函数updatePoints的所有代码来说都是可见的。

bonus和newPoints也是函数updatePoints的局部变量，而userPoints是全局变量。

在这些代码中，只能使用全局变量，无法访问函数中定义的任何局部变量，因为局部变量在全局作用域内不可见。



短命的变量

如果你是变量，就努力拼搏吧，因为你的日子可能不多了。除非你是全局变量，否则情况就是这样的，但即便是全局变量，寿命也是有限的。是什么决定了变量的寿命呢？你可以这么认为。

全局变量的寿命与网页一样长。全局变量在JavaScript代码加载到网页之后降生，并在网页消失后死去。重新加载网页时，将销毁并重新创建所有的全局变量。

局部变量通常在函数结束时消失。局部变量是在函数被调用后创建的，并一直活到函数返回（无论函数是否返回值）。然而，在局部变量的大限到来前，可从函数返回它们的值。

这里说“通常”是因为有一些高级技巧，可稍微延长局部变量的寿命，但我们暂时不考虑这些。

从上面的介绍可知，变量的寿命不可能超过网页的寿命。如果你是局部变量，将来去匆匆；即便你足够幸运，是全局变量，最多也只能活到浏览器重新加载网页之时。

千万别忘了声明局部变量

如果你使用未声明的变量，它就会是全局的。这意味着即便你首次使用一个变量时是在函数内部（因为你想将其作为局部变量），这个变量也将是全局的，在函数外面也可用（这可能带来麻烦）。因此，对于局部变量，千万别忘了声明它们！

使用未声明的变量时，它将自动被视为全局变量，即便你在函数中首次使用它亦如此。

```
function playTurn(player, location) {
    points = 0;
    if (location == 1) {
       points = points + 100;
    }
    return points;
}
var total = playTurn("Jai", 1);
alert(points);
```

使用变量points前，我们忘了使用关键字var声明它，因此它被自动视为全局变量。

这意味着可以在函数外面使用points！函数执行完毕后，这个变量并不会像你希望的那样消失。

上述代码与下面的代码等价：

由于你忘记使用关键字var，JavaScript认为你要将points作为全局变量，就像在全局作用域内声明了它一样。

```
var points = 0;
function playTurn(player, location) {
    points = 0;
    if (location == 1) {
       points = points + 100;
    }
    return points;
}
var total = playTurn("Jai", 1);
alert(points);
```

忘记声明局部变量可能会带来麻烦——如果使用了同名的全局变量，这可能会修改并非你要修改的值。





它将“遮住”（shadow）全局变量。

这是什么意思呢？假设你定义了全局变量**beanCounter**，然后声明了如下函数：

```
var beanCounter = 10;

function getNumberOfItems(ordertype) {
    var beanCounter = 0;
    if (ordertype == "order") {
       // 对bean Counter做些什么
    }
    return beanCounter;
}
```

有一个名为*beanCounter*的全局变量，还有一个名为*beanCounter*的局部变量。

如果你这样做，每次在函数中使用**beanCounter**时，引用的都是局部变量，而不是全局变量。因此，我们说局部变量遮住了全局变量。换句话说，我们看不到全局变量，因为它被局部变量挡住了。

请注意，全局变量和局部变量不会相互影响：如果你修改其中的一个，对另一个不会有任何影响。它们是彼此独立的变量。



下面的JavaScript代码，包含一些变量、函数定义和函数调用。你的任务是找出所有的实参、形参、局部变量和全局变量，将它们填在右边相应的方框中，并圈出被遮住的变量。记得查看本章末尾的答案。

```
var x = 32;
var y = 44;
var radius = 5;

var centerX = 0;
var centerY = 0;
var width = 600;
var height = 400;

function setup(width, height) {
    centerX = width/2;
    centerY = height/2;
}

function computeDistance(x1, y1, x2, y2) {
    var dx = x1 - x2;
    var dy = y1 - y2;
    var d2 = (dx * dx) + (dy * dy);
    var d = Math.sqrt(d2);
    return d;
}

function circleArea(r) {
    var area = Math.PI * r * r;
    return area;
}

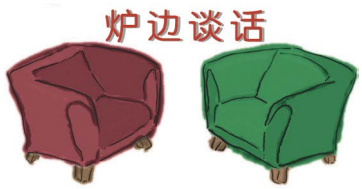
setup(width, height);
var area = circleArea(radius);
var distance = computeDistance(x, y, centerX, centerY);
alert("Area: " + area);
alert("Distance: " + distance);
```

实参

形参

局部变量

全局变量



今晚主题：**全局变量和局部变量争论谁对程序来说更重要。**

全局变量

嘿，局部变量，我真不知道为什么要用你，因为我能满足程序员的所有变量需求。毕竟，我在什么地方都是可见的！

你必须承认，可将所有的局部变量替换为全局变量，同时确保函数完全像以前一样工作。

不一定会乱七八糟。程序员可在程序开头创建需要的所有变量，这样变量就都在一个地方了。

只要命名得当，就能够更轻松地记住变量。

确实是这样。但如果需要的所有值都存储在全局变量中，还需要实参和形参做什么呢？

局部变量

没错，但到处使用全局变量绝对是糟糕的编程风格。很多函数都需要局部变量，即只有它们自己能用的私有变量，而全局变量在什么地方都是可见的。

你这话说对也对，说不对也不对。如果特别小心，确实如此。但要做到那么谨慎很难，一不小心就会让函数使用其他函数用于其他目的的变量。这还会让程序乱七八糟，充斥着只有一个函数需要的全局变量。

调用需要变量（如x）的函数时，如果程序员不记得x以前被用于做什么，结果将如何呢？程序员必须查看所有的代码，看看是否在其他地方使用过x！这真是噩梦。

那形参呢？函数形参都是局部变量，就是想避也避不开。

看你说的。函数的根本目标在于让程序员能够重用代码，以便根据不同的输入计算不同的东西。

全局变量

但局部变量的寿命都很短，如昙花一现。

根本不使用全局变量？对JavaScript程序员来说，全局变量就是中流砥柱！

看来我得喝一杯了。

局部变量

面对现实吧。除非万不得已，否则就应该使用局部变量而不是全局变量，这是一种良好的编程实践。全局变量会带来大麻烦。我见过一些JavaScript程序基本上不使用全局变量！

对没有经验的程序员来说，确实如此。但程序员在学会如何正确地组织代码，以确保正确性、提高可维护性和遵循良好的编码风格后，就只会在迫不得已的情况下使用全局变量。

让全局变量喝一杯？看来这个世界真是太危险了。



世上没有愚蠢的问题

问：记住所有局部变量和全局变量的作用域很难，为何不始终使用全局变量？我一直都是这样做的。

答：如果你要编写的代码很复杂或需要长期维护，就必须小心地管理变量。如果大量创建全局变量，将难以跟踪都在什么地方使用了这些变量（以及在什么地方修改了它们的值），而这可能导致代码充斥bug。与人合作编写代码或使用第三方库时，避免这些问题就更为重要（编写良好的第三方库也会尽量避免这些问题）。

因此，可在合适的情况下使用全局变量，但要适度，并尽可能使用局部变量。获得更多JavaScript经验后，你就能使用其他代码组织技术，让代码更容易维护。

问：我在网页中使用了一些全局变量，同时加载了其他JavaScript文件。请问这些文件的全局变量是独立的吗？

答：只有一个全局作用域，因此加载的每个文件看到的变量都相同（它们创建的全局变量位于同一个空间内）。因此，避免使用的变量发生冲突（并尽可能少用甚至不用全局变量）至关重要。

问：如果形参与全局变量同名，它会遮住全局变量吗？

答：会。与在函数中声明了与全局变量同名的局部变量一样，如果使用了与全局变量同名的形参，该形参也会遮住相应的全局变量。只要在函数中不使用该全局变量，遮住它就没有什么关系，但最好使用注释对这一点进行说明，以免以后阅读代码时感到迷惑。

问：在浏览器中重新加载网页时，会重新初始化所有的全局变量吗？

答：是的。对变量而言，重新加载网页与首次加载网页一样。如果重新加载网页时正在执行代码，所有的局部变量也都会消失。

问：局部变量必须在函数开头声明吗？

答：与全局变量一样，可在函数中首次需要局部变量时声明它。然而，在函数开头声明局部变量是一种良好的编程实践，这样别人阅读你的代码时就能很容易找到这些声明，看一眼就知道函数使用了哪些变量。另外，推迟声明变量时，如果在函数体中原来预期的位置之前使用了它们，将可能导致意料之外的行为。JavaScript在函数开始执行时创建所有的局部变量，而不管这些变量是否已经声明（这被称为提升，后面将更详细地讨论），但变量在赋值前都是未定义的，这可能不符合你的初衷。

问：所有人都抱怨JavaScript过度使用全局变量。为什么会这样？是这门语言设计得太糟糕、大家没有意识到自己在过度使用全局变量，还是其他什么原因呢？我们该怎么办？

答：在JavaScript中，常常过度使用全局变量。部分原因是JavaScript对代码的结构要求不严，不需要规划就可直接编写代码，这其实是件好事。对于需要长期维护和修改的重要代码（所有的网页都属于这样的代码），这样编写的缺点就暴露出来了。虽然有这样或那样的缺点，但JavaScript是一门功能强大的语言，提供了对象等特性，让你能够以模块化的方式组织代码。很多图书都专门介绍了对象这一主题，本书第5章也会让你对其有大致的了解。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权



实际上，可将函数放在JavaScript文件的任何地方。

JavaScript不在乎函数是在使用前还是使用后声明的。例如，请看下面的代码：

```
var radius = 5;
var area = circleArea(radius);
alert(area);

function circleArea(r) {
    var a = Math.PI * r * r;
    return a;
}
```

注意到我们在定义函数circleArea前使用了它！

在上述代码中，函数circleArea实际上是在调用后才定义的。这到底是怎么回事呢？

这好像确实很奇怪，如果你还记得浏览器加载网页时就开始按从上到下的顺序执行文件的JavaScript代码，就更会觉得奇怪。然而，实际情况是，浏览器分两遍读取网页：第一遍读取所有的函数定义，第二遍开始执行代码。这让你可以将函数放在文件的任何地方。



无名装置

无名装置（Thing-A-Ma-Jig）设计得非常巧妙，它能够发出叮当声（clank）、沉闷的金属声（clunk）和榔榔声（thunk）。它到底是做什么的呢？我们也说不清，但程序员宣称他们知道其中的工作原理。你能破解这些代码，找出其中的奥秘吗？

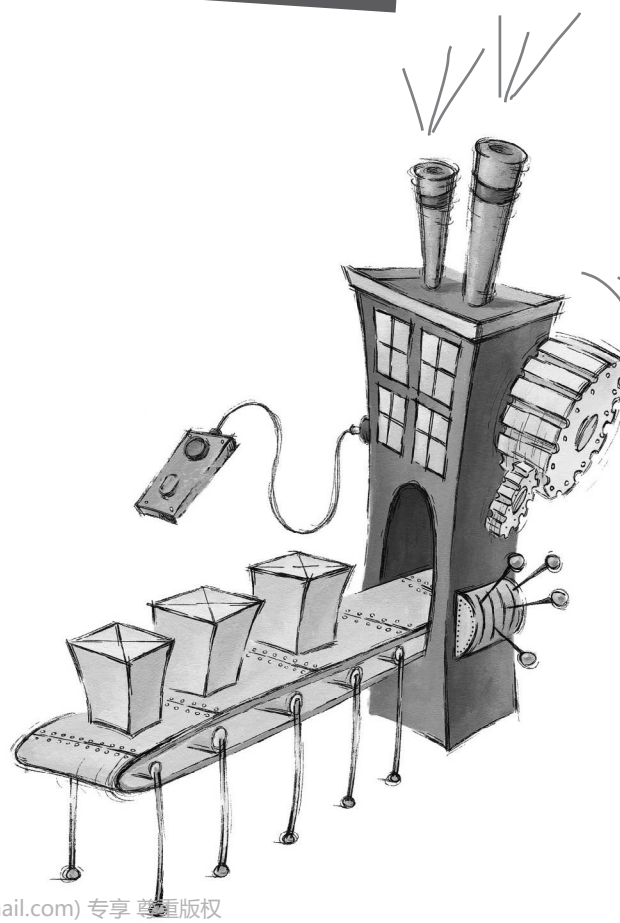
```
function clunk(times) {
    var num = times;
    while (num > 0) {
        display("clunk");
        num = num - 1;
    }
}

function thingamajig(size) {
    var facky = 1;
    clunkCounter = 0;
    if (size == 0) {
       display("clank");
    } else if (size == 1) {
       display("thunk");
    } else {
        while (size > 1) {
            facky = facky * size;
            size = size - 1;
        }
        clunk(facky);
    }
}

function display(output) {
    console.log(output);
    clunkCounter = clunkCounter + 1;
}
var clunkCounter = 0;
thingamajig(5);
console.log(clunkCounter);
```

在这里列出你认为的输出！

JavaScript控制台



建议你向这台无名装置输入0、1、2、3、4、5等，看看能否明白它是做什么的。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 禁止重版权

Web镇代码卫生指南



在Web镇，大家喜欢保持整洁有序，为扩展做好准备。说到保持整洁有序，代码对这种需求十分迫切，但在组织变量和函数方面，JavaScript的要求太过宽松。有鉴于此，我们编写了这份简要的指南，向Web镇的新人提出了一些建议。该指南免费提供，拿一份吧。

在开头声明全局变量！

尽可能将全局变量都放在一起；如果将它们都放在开头，就很容易找到。你并非必须这样做，但如果将代码中用到的变量都放在开头，是不是更容易找到呢？

函数喜欢扎堆。

不完全是这样，它们实际上并不在乎，因为它们只是函数。但如果将函数放在一起，查找起来将容易得多。你知道，浏览器实际上首先在JavaScript代码中找出函数。因此你可以将函数放在文件开头，也可以放在文件末尾，但如果将它们都放在一个地方，你的工作将更轻松。在Web镇，大家通常将全局变量放在开头，再接着定义函数。

在函数开头声明全局变量。

将所有的局部变量声明都放在函数体开头，这样更容易找到它们，还能确保使用前正确地声明了它们。

就这些，愿你享受在Web镇的编码时光。



猜猜我是谁



一大拨JavaScript成员乔装打扮，正在玩一个“猜猜我是谁”的晚会游戏。你需要根据它们对自己的描述猜测它们的身份——假设它们描述自己时说的都是真话。请在下面的空白处填写每位成员的名字。

参与今晚游戏的成员有：

`function`、实参、`return`语句、作用域、局部变量、全局变量、按值传递、形参、函数调用、`Math.random`、内置函数、代码重用。

我被传递给函数。 _____

我向调用函数的代码返回值。 _____

我是最重要的关键字。 _____

我获得实参的值。 _____

我意味着创建副本。 _____

我在什么地方都可见。 _____

我是调用函数的近义词。 _____

我是一个与对象相关联的函数。 _____

我是`alert`和`prompt`所属的函数类别。 _____

我是函数最擅长实现的功能。 _____

我表示变量在什么地方可见。 _____

我所属的函数在，我就在。 _____

那个盗窃未遂案没必要调查

给成事不足的雷斯垂德警长打完电话，福尔摩斯在壁炉前坐下，接着看报纸。华生充满期待地看着他。



福尔摩斯头也不抬地说：“干什么？”

华生问：“雷斯垂德怎么说？”

“哦，他说他在银行账户中找到了可疑的流氓代码。”

“还有呢？”华生极力地掩饰自己的失望。

福尔摩斯说：“雷斯垂德通过邮件把代码发给了我，我跟他说这个案件不用查了。罪犯犯了一个致命的错误，根本不可能把钱偷走。”

“你是怎么知道的？”华生问。

“要是你懂，这显而易见，”福尔摩斯气愤地说，“别再问我了，让我把这张报纸看完。”

福尔摩斯全神贯注地看着报纸，华生悄悄地瞄了一眼福尔摩斯的手机，打开雷斯垂德发来的邮件，并研究其中的代码。

这是银行账户的真正实际余额。

```
var balance = 10500;
var cameraOn = true;

function steal(balance, amount) {
    cameraOn = false;
    if (amount < balance) {
        balance = balance - amount;
    }
    return amount;
    cameraOn = true;
}

var amount = steal(balance, 1250);
alert("Criminal: you stole " + amount + "!");
```

福尔摩斯为何决定不调查这个案件？他怎么看一眼代码就知道罪犯根本就不可能把钱偷走？这些代码有什么问题吗？



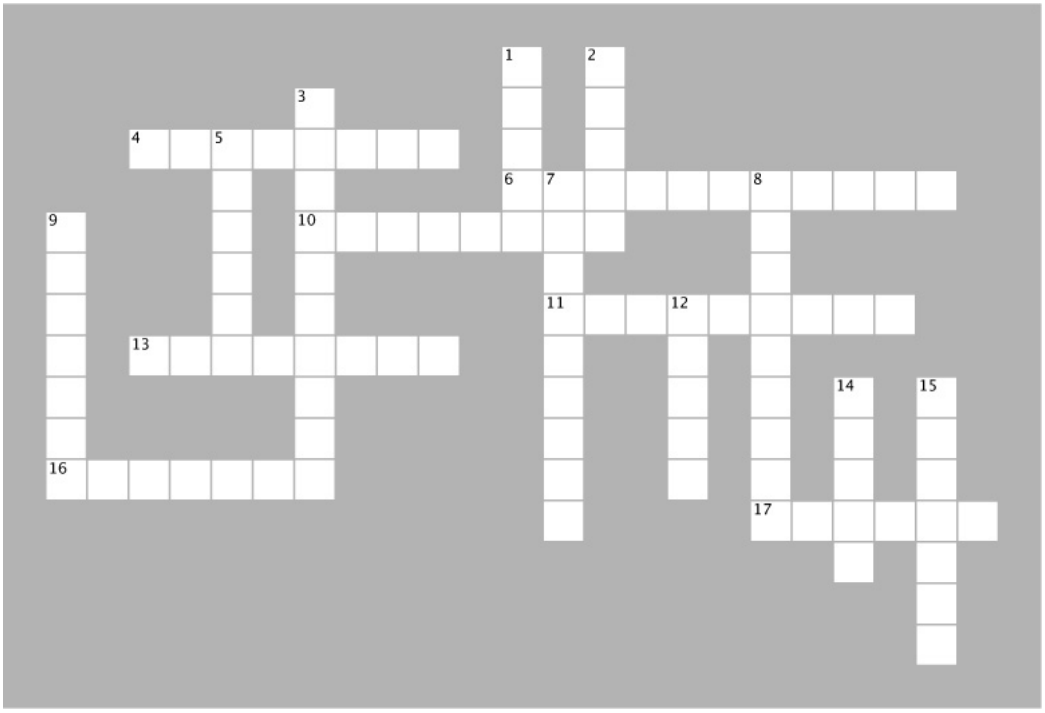
要点

- 要声明函数，使用关键字function，并在它后面指定函数名。
- 将函数的所有**形参**都放在圆括号内；如果没有形参，就使用空圆括号。
- 将**函数体**放在花括号内。
- 函数体中的语句在你调用函数时执行。
- 要调用函数，指定函数名，并将实参传递给函数的形参（如果有形参的话）。
- 函数可使用return语句返回一个值。
- 函数为形参创建局部变量，并创建它需要使用的局部变量。
- 变量的作用域要么是**全局**的（在程序的任何地方都可见），要么是**局部**的（只在声明它的函数中可见）。
- 在函数体开头声明局部变量。
- 如果忘记使用var声明局部变量，它将是全局变量，这可能给程序带来意外的影响。
- 函数是一种很好的代码组织方式，它创建可重用的代码块。
- 可通过将实参传递给形参来自定义函数代码，还可使用不同的实参来获得不同的结果。
- 函数是一种减少乃至消除重复代码的极佳方式。
- 在程序中，可使用很多JavaScript内置函数（如alert、prompt和Math.random）来完成工作。
- 使用内置函数意味着使用既有代码，不需要你自己编写。
- 最好这样组织代码：将函数放在一起，并将所有全局变量都放在JavaScript文件的开头。



JavaScript填字游戏

在本章中，你养成了函数思维。现在请开动你的脑筋，完成下面的填字游戏。



横向

4. 在函数体内，形参就像什么一样？

6. JavaScript向函数传递实参时采用的做法。

10. 可在JavaScript文件的哪些地方声明函数？

11. 没有`return`语句的函数返回的值。

13. 局部变量在什么返回时消失？

16. 如果你忘记了声明局部变量，它将被视为什么（复数形式）？

17. 局部变量会如何影响同名的全局变量？

纵向

1. 在任何地方都可见的变量拥有全局什么？

2. 使用函数可反复地对代码进行什么？

3. 传递给函数的实参将赋给的变量。

5. 用于从函数返回值的语句。

7. 传递给函数的东西。

8. 重新加载网页时，都将被重新初始化的东西。

9. 观察代码的逐行执行过程。

12. 华生通过福尔摩斯的手机中的什么查看了银行盗窃代码？

14. 应尽可能使用的变量类型。

15. 多余的实参将被函数怎么样？



磨笔上阵 答案

请简单地分析下面的代码。它给你什么样的感觉？请选择后面你认为合适的评价，你也可以写下自己的评价。答案如下。

```
var dogName = "rover";
var dogWeight = 23;
if (dogWeight > 20) {
   console.log(dogName + " says WOOF WOOF");
} else {
   console.log(dogName + " says woof woof");
}
dogName = "spot";
dogWeight = 13;
if (dogWeight > 20) {
   console.log(dogName + " says WOOF WOOF");
} else {
   console.log(dogName + " says woof woof");
}
dogName = "spike";
dogWeight = 53;
if (dogWeight > 20) {
   console.log(dogName + " says WOOF WOOF");
} else {
   console.log(dogName + " says woof woof");
}
dogName = "lady";
dogWeight = 17;
if (dogWeight > 20) {
   console.log(dogName + " says WOOF WOOF");
} else {
   console.log(dogName + " says woof woof");
}
```

我们选择所有的评价。

- ☑ A. 重复的代码多得不得了。
- ☑ B. 如果要修改输出的显示方式或添加其他小狗属性，将需要做大量重复的工作。
- ☑ C. 输入起来很繁琐！
- ☑ D. 可读性不佳。
- ☑ E. 看起来这位开发人员认为体重可能随时间变化。



下面是一些调用函数bark的代码。在每个调用旁边指出该调用是否会导致错误；如果不会，就写出其输出。答案如下。

```
bark("juno", 20);
```
juno says woof woof

```
bark("scottie", -1);
```
scottie says woof woof

bark函数没有通过检查来确认小狗的体重大于零；因此该调用显示上面的输出，因为–1小于20。

```
bark("dino", 0, 0);
```
dino says woof woof

函数bark忽略多余的参数0。体重为零不合理，但这个函数调用不会导致问题。

```
bark("fido", "20");
```
fido says woof woof

函数bark将字符串“20”与数字20进行比较，它不比20大，因此显示fido says woof woof。（本书后面将介绍JavaScript如何对“20”和20进行比较。）

```
bark("lady", 10);
```
lady says woof woof

```
bark("bruno", 21);
```
bruno says WOOF WOOF



代码冰箱贴答案

冰箱上贴着一些被打乱的JavaScript代码。你能将这些代码重组为一个生成如下输出的程序吗？请注意，冰箱上的有些代码可能是多余的，也就是说你可能不会用到所有的冰箱贴。答案如下。

```
function whatShallIWear ( temp ) {
  if (temp < 60) {
    console.log("Wear a jacket");
  }
  else if (temp < 70) {
    console.log("Wear a sweater");
  }
  else {
    console.log("Wear t-shirt");
  }
}
whatShallIWear(50);
whatShallIWear(80);
whatShallIWear(60);
```

多出来的冰箱贴

```
,
,
,
}
}
temperature
```

JavaScript控制台

```
Wear a jacket
Wear a t-shirt
Wear a sweater
```


图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权



下面是一些JavaScript代码，其中包含变量、函数定义和函数调用。你的任务是找出所有的变量、函数、实参和形参，并将它们填在右边相应的方框中。答案如下。

```
function dogYears(dogName, age) {
    var years = age * 7;
    console.log(dogName + " is " + years + " years old");
}
var myDog = "Fido";
dogYears(myDog, 4);

function makeTea(cups, tea) {
    console.log("Brewing " + cups + " cups of " + tea);
}
var guests = 3;
makeTea(guests, "Earl Grey");

function secret() {
    console.log("The secret of life is 42");
}
secret();

function speak(kind) {
    var defaultSound = "";
    if (kind == "dog") {
        alert("Woof");
    } else if (kind == "cat") {
        alert("Meow");
    } else {
        alert(defaultSound);
    }
}
var pet = prompt("Enter a type of pet: ");
speak(pet);
```

变量

myDog, guests, pet, years, defaultSound

函数

dogYears, makeTea, secret, speak,

内置函数

alert, console.log, prompt

实参

myDog, 4, guests, “Earl Grey”, pet, plus all the string arguments to alert and console.log

形参

dogName, age, cups, tea, kind



下面的JavaScript代码包含一些变量、函数定义和函数调用。你的任务是找出所有的实参、形参、局部变量和全局变量，将它们填在右边相应的方框中，并圈出被遮住的变量。答案如下。

```
var x = 32;
var y = 44;
var radius = 5;

var centerX = 0;
var centerY = 0;
var width = 600;
var height = 400;

function setup(width, height) {
    centerX = width/2;
    centerY = height/2;
}

function computeDistance(x1, y1, x2, y2) {
    var dx = x1 - x2;
    var dy = y1 - y2;
    var d2 = (dx * dx) + (dy * dy);
    var d = Math.sqrt(d2);
    return d;
}

function circleArea(r) {
    var area = Math.PI * r * r;
    return area;
}

setup(width, height);
var area = circleArea(radius);
var distance = computeDistance(x, y, centerX, centerY);
alert("Area: " + area);
alert("Distance: " + distance);
```

实参

width, height, radius,
x, y, centerX, centerY,
"Area: " + area,
"Distance: " +
distance

别忘了调用函数alert时传入的实参。

形参

width, height, x1,
y1, x2, y2, r

局部变量

dx, dy, d2, d,
area

局部变量area遮住了全局变量area。

全局变量

x, y, radius, centerX,
centerY, width,
height, area,
distance

别忘了area和distance，它们也是全局变量。



无名装置

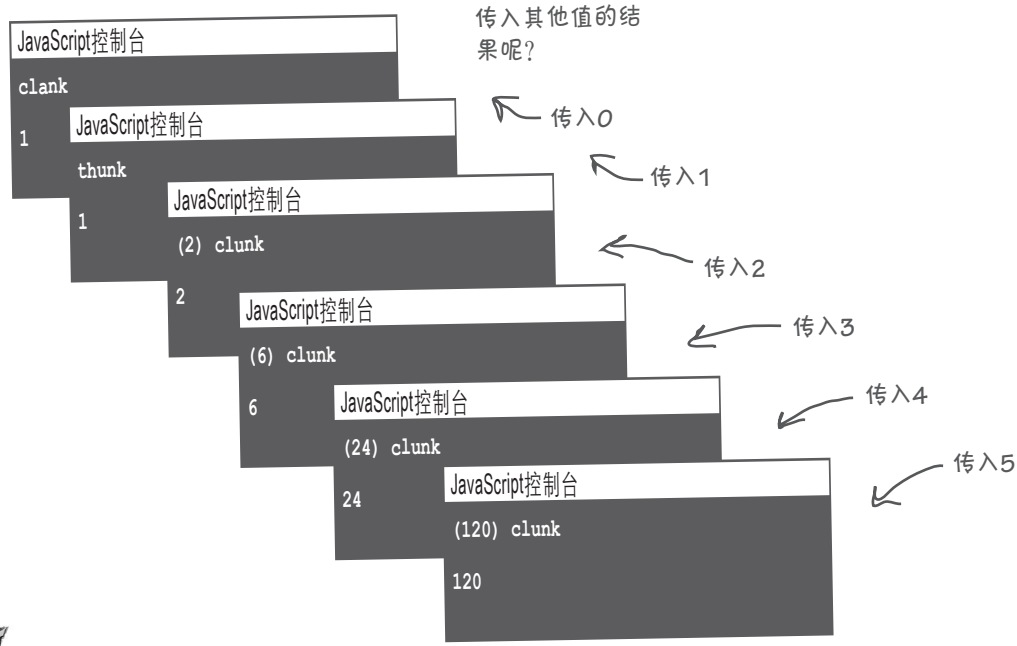
无名装置（Thing-A-Ma-Jig）设计得非常巧妙，它能够发出叮当声（clank）、沉闷的金属声（clunk）和榔榔声（thunk）。它到底是做什么的呢？我们也说不清，但程序员宣称他们知道其中的工作原理。你能破解这些代码，找出其中的奥秘吗？

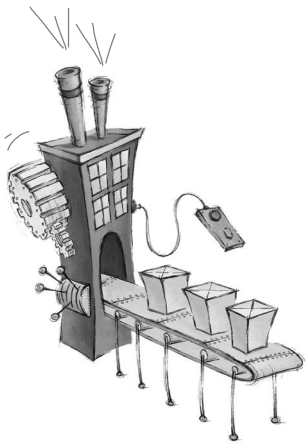
答案如下。



如果给函数thingamajig传递5，控制台将显示120次clunk（控制台可能简写为(120) clunk，如上所示），并在最后显示数字120。

传入其他值的结果呢？





这些结果到底意味着什么呢？据说无名装置是好奇心极强的家伙发明的，他对重新排列单词着迷。你知道，对于单词DOG，可重新排列为GOD、OGD、DGO、GDO和ODG。因此，如果单词包含三个字母，无名装置将指出这些字母总共有6种不同的组合。对于单词mixes，其中的字母有120种不同的组合，真多！这些都是传闻，我们在这里认为这种无名装置计算的是数学上的阶乘！谁知道呢！

搜索“阶乘”了解更多信息！

5分钟侦探小说答案



福尔摩斯为何决定不调查这个案件？他怎么看一眼代码就知道罪犯根本就不可能把钱偷走？这些代码有什么问题吗？答案如下。

```
var balance = 10500;
var cameraOn = true;

function steal(balance, amount) {
    cameraOn = false;
    if (amount < balance) {
        balance = balance - amount;
    }
    return amount;
    cameraOn = true;
}

var amount = steal(balance, 1250);
alert("Criminal: you stole " + amount + "!");
```

balance是一个全局变量。

在函数steal中，它被这个形参遮住了。

因此，在函数steal中修改balance时，并不会改变银行账户的实际余额。

返回窃取的金额。

但没有根据它来修改账户的实际余额，因此银行账户的实际余额与原来一样。

罪犯以为得逞了，可实际上没有。

罪犯不但没有得逞，还忘记了将监控重新打开，给警察留下了有人实施犯罪的铁证。别忘了，从函数返回时，函数将停止执行，因此return语句后面的所有代码都将被忽略！

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

猜猜我是谁



答案

一大拨JavaScript成员乔装打扮，正在玩一个“猜猜我是谁”的晚会游戏。你需要根据它们对自己的描述猜测它们的身份——假设它们描述自己时说的都是真话。请在下面的空白处填写每位成员的名字。

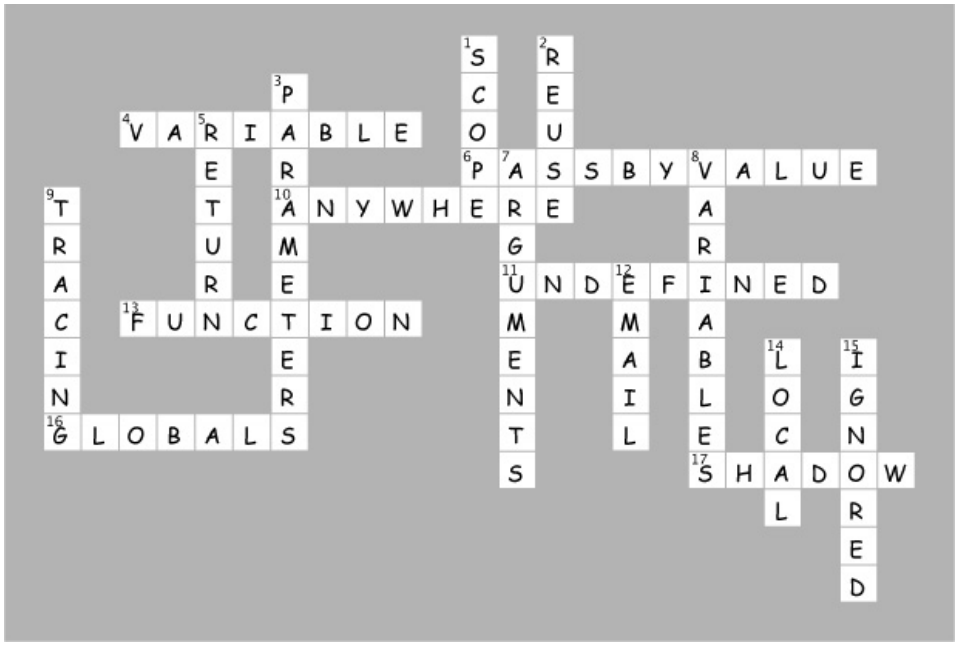
参与今晚游戏的成员有：

`function`、实参、`return`语句、作用域、局部变量、全局变量、按值传递、形参、函数调用、`Math.random`、内置函数、代码重用。

我被传递给函数。	实参
我向调用函数的代码返回值。	return语句
我是最重要的关键字。	function
我获得实参的值。	形参
我意味着创建副本。	按值传递
我在什么地方都可见。	全局变量
我是调用函数的近义词。	函数调用
我是一个与对象相关联的函数。	Math.random
我是`alert`和`prompt`所属的函数类别。	内置函数
我是函数最擅长实现的功能。	代码重用
我表示变量在什么地方可见。	作用域
我所属的函数在，我就在。	局部变量



JavaScript填字游戏答案



4 让数据排排坐

数组

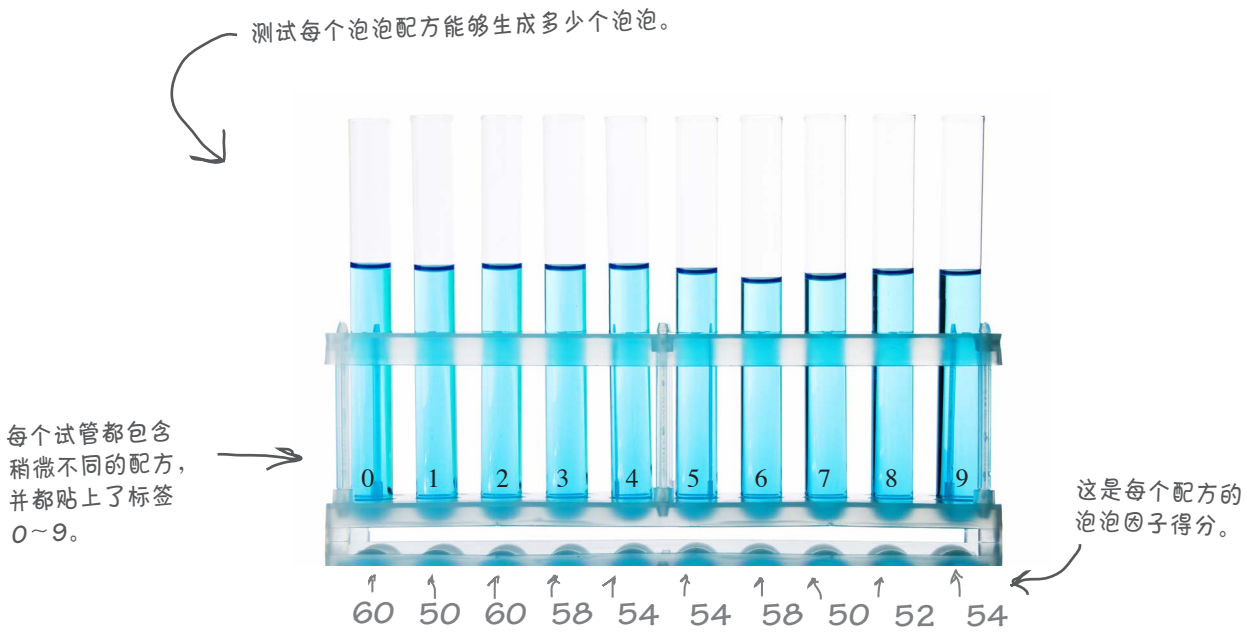


在JavaScript中，并非只有数字、字符串和布尔值。前面编写JavaScript代码时，使用的都是**基本类型**（简单字符串、数字和布尔值），如`"Fido"`、`23`和`true`。使用基本类型可完成很多工作，但有时候必须处理**更多数据**，如购物车中的所有商品、播放列表中的所有歌曲、一系列恒星及其亮度或整个产品目录，为此，需要更强大的工具。对于这种按顺序排列的数据，可使用JavaScript**数组**来存储。本章将介绍如何将数据加入数组、如何传递数组以及如何操作数组。本书后面将介绍其他**组织数据**的方式，但现在先从数组着手吧。



你能帮助泡泡玩具反斗城公司吗

来认识一下泡泡玩具反斗城（Bubbles-R-Us）公司。这家公司孜孜不倦地研究，旨在让泡泡魔棒和泡泡机能够吹出最好的泡泡。今天，公司要对多个新配方的“泡泡因子”进行测试，确定各个配方都能生成多少个泡泡。下面是得到的测试数据。



当然，你应该将这些数据都输入JavaScript，以便能够编写代码来进行分析。但数据很多，如何编写处理这些数据的代码呢？

如何使用JavaScript表示多个值

你知道如何使用JavaScript来表示单个值，如字符串、数字和布尔值，但如何表示**多个值**（如10个配方的泡泡因子得分）呢？使用JavaScript**数组**。数组是一种可存储很多值的JavaScript数据类型。下面的JavaScript数组存储了所有的泡泡因子得分：

```
var scores = [60, 50, 60, 58, 54, 54, 58, 50, 52, 54];
```

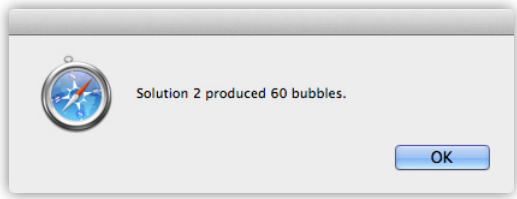
这里总共有10个值，它们组合在一起，存储在一个数组中，并被赋给变量scores。

可将所有这些值视为一个整体，也可在需要时访问其中的各个值，如下所示。

要访问数组的元素，我们使用这样的语法：数组变量名加上用方括号括起的元素索引。

请注意，数组的索引从0开始。因此，第一个配方为0号配方，其得分存储在scores[0]中；同理，第三个配方为2号配方，其得分存储在scores[2]中。

```
var solution2 = scores[2];
alert("Solution 2 produced " + solution2 + " bubbles.");
```





泡泡玩具反斗城公司的一位泡泡学家。

数组的工作原理



向泡泡玩具反斗城公司伸出援手前，咱们先把数组弄明白。前面说过，可使用数组来存储**多个**值（而不像变量那样只能存储一个值，如一个数字或一个字符串）。当需要将类似的东西（如泡泡因子得分、冰淇淋口味、白天的温度或一组判断题的答案）编组时，通常使用数组。有了一系列要编组的值后，就可创建一个数组来存储它们，并在需要时访问数组中的这些值。

如何创建数组

假设你要创建一个存储冰淇淋口味的数组，可以这样做：

注意到使用了逗号来分隔数组的各个元素。

```
var flavors = ["vanilla", "butterscotch", "lavender", "chocolate", "cookie dough"];
```

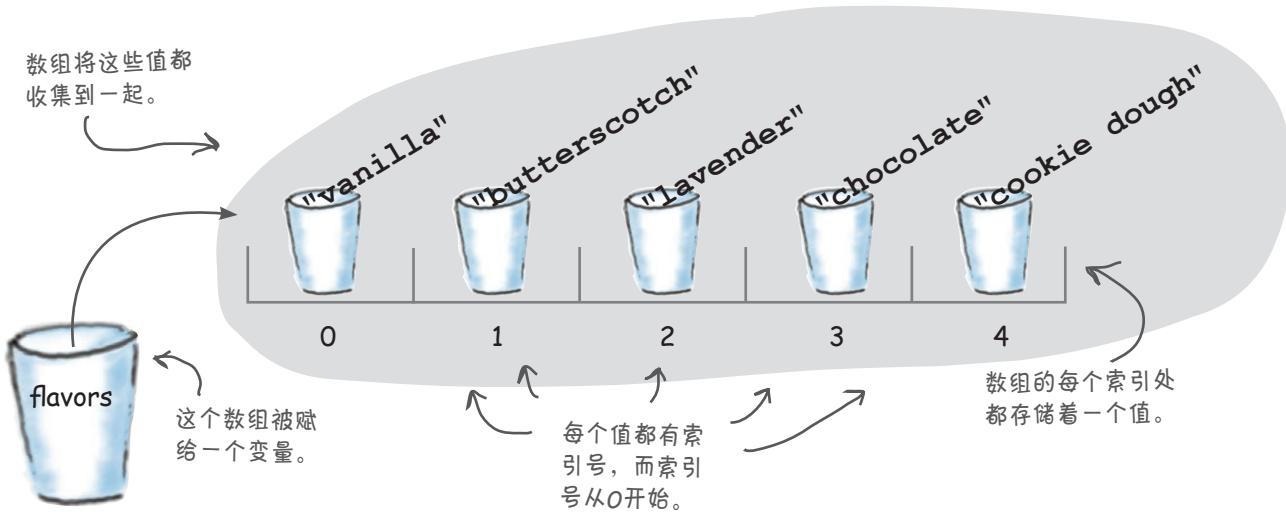
将这个数组赋给变量flavors。

使用字符[指定数组的开始位置。

接下来列出所有的数组元素。

使用字符]指定数组的结束位置。

创建数组时，每个元素都放在特定的位置（**索引**，index）。在数组`flavors`中，第一个元素`"vanilla"`位于索引0处，第二个元素`"butterscotch"`位于索引1处，依此类推。下面是一种看待数组的方式。



如何访问数组元素

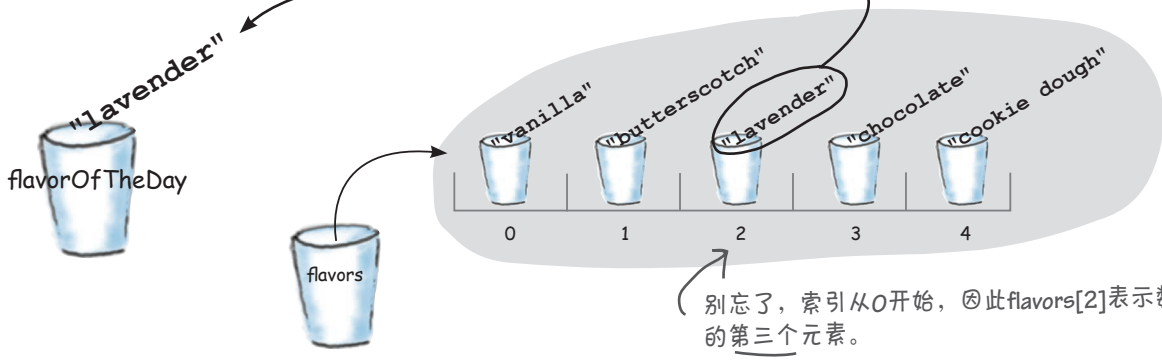
数组的每个元素都有索引，这是你访问和修改数组中值的钥匙。要访问元素，只需在数组变量名后面加上用方括号括起的索引；在可以使用变量的任何地方，都可使用这种表示法：

```
var flavorOfTheDay = flavors[2];
```

它表示数组flavors的索引2处的值（即“lavender”），将这个值赋给了变量flavorOfTheDay。

要获取数组中的元素，需要使用数组名以及元素的索引。

将flavors[2]中的值赋给flavorOfTheDay。



别忘了，索引从0开始，因此flavors[2]表示数组的第三个元素。

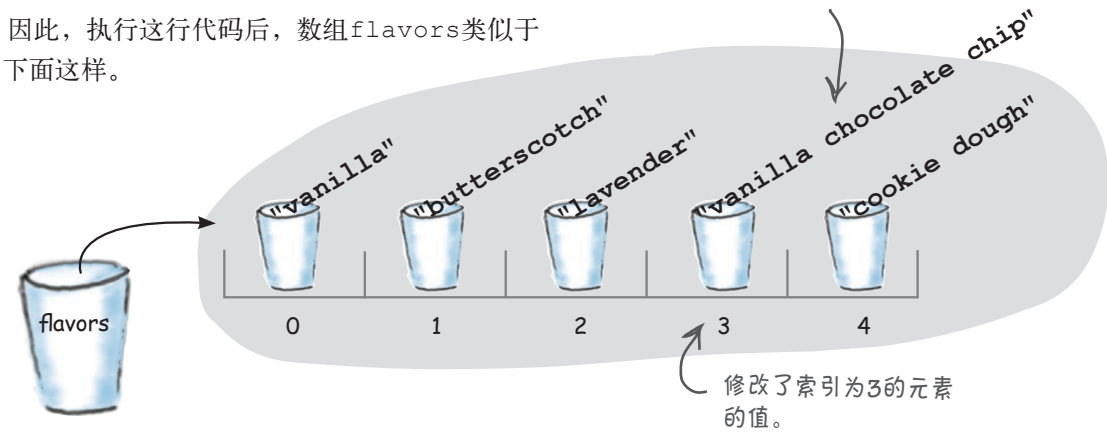
修改数组元素

还可使用数组索引来修改数组中的值：

```
flavors[3] = "vanilla chocolate chip";
```

将索引3处的值设置为新值“vanilla chocolate chip”（原来为“chocolate”）。

因此，执行这行代码后，数组flavors类似于下面这样。



修改了索引为3的元素的值。

确定数组的长度

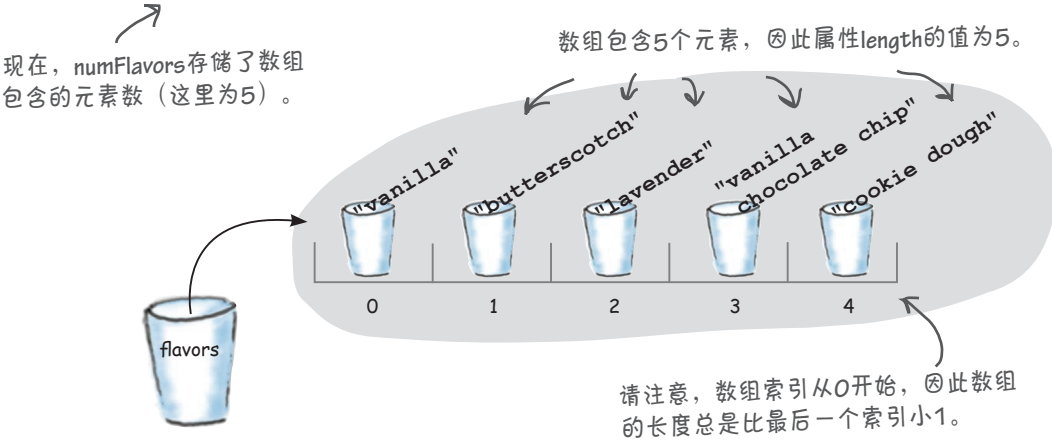
假设有人给了你一个非常大的数组，其中包含重要的数据。你知道它包含什么内容，但可能不知道它到底多长。所幸每个数组都有属性length。第5章将更详细地讨论属性及其工作原理，但就现在而言，你只需知道属性是一个与数组相关联的值即可。下面演示了如何使用属性length。

每个数组都有属性length，指出了数组当前包含多少个元素。

要获取数组的长度，可使用数组名、句点和length。

```
var numFlavors = flavors.length;
```

现在，numFlavors存储了数组包含的元素数（这里为5）。



磨笔上阵

下面的products数组存储了Jenn & Berry销售的各种冰淇淋口味。这些冰淇淋口味是按制作顺序添加到数组中的，请补全下面的代码，找出最后制作的冰淇淋口味。

```
var products = ["Choo Choo Chocolate", "Icy Mint", "Cake Batter", "Bubblegum"];
var last = ____________________;
var recent = products[last];
```



图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

试试我新推出的自动造句应用程序吧，你将像老板和市场营销人员一样巧舌如簧。



你认为第1章的商务应用程序不够正式？如果你要向老板展示一流的应用程序的话，试试这个吧。

下面是炙手可热的自动造句（Phrase-o-Matic）应用程序的代码，你能明白它是做什么的吗？

```
<!doctype html>
<html lang="en">
<head>
  <title>Phrase-o-matic</title>
  <meta charset="utf-8">
  <script>
    function makePhrases() {
      var words1 = ["24/7", "multi-tier", "30,000 foot", "B-to-B", "win-win"];
      var words2 = ["empowered", "value-added", "oriented", "focused", "aligned"];
      var words3 = ["process", "solution", "tipping-point", "strategy", "vision"];

      var rand1 = Math.floor(Math.random() * words1.length);
      var rand2 = Math.floor(Math.random() * words2.length);
      var rand3 = Math.floor(Math.random() * words3.length);

      var phrase = words1[rand1] + " " + words2[rand2] + " " + words3[rand3];
      alert(phrase);
    }
    makePhrases();
  </script>
</head>
<body></body>
</html>
```

自动造句应用程序

但愿你已经明白了，这些代码是为创业公司推出下一个广告语的绝佳工具。它曾推出了成功的广告语Win-win value-added solution和24/7 empowered process，未来极有可能推出其他成功的广告语。来看看这些代码的工作原理吧。

1. 首先，我们定义了函数makePhrases。我们可根据需要调用这个函数任意多次，以造出想要的句子。

```
                                    定义可在后面调用的函数makePhrases。
function makePhrases() {
                                    makePhrases的代码都放在这里，我们稍后将介绍它们。
}
makePhrases();                      调用函数makePhrases一次，但如
                                    果需要多个句子，可调用它多次。
```

2. 现在可以编写函数makePhrases的代码了。首先创建三个数组，每个都存储了用来造句的单词。在下一步中，我们将从每个数组中随机选择一个单词，以造出一个包含三个单词的句子。

```
创建变量words1，用于存储第一个数组。
var words1 = ["24/7", "multi-tier", "30,000 foot", "B-to-B", "win-win"];
                                    在这个数组中添加5个字符串。
                                    你可将它们换成最时髦的词语。

var words2 = ["empowered", "value-added", "oriented", "focused", "aligned"];
var words3 = ["process", "solution", "tipping-point", "strategy", "vision"];
                                    这是另外两个包含单词的数组，将
                                    它们赋给了变量words2和words3。
```

③ 生成三个随机数，以随机地选择三个用于造句的单词。第2章说过，`Math.random`生成一个0～1（不包括1）的数字。通过将其乘以数组的长度，并使用`Math.floor`将结果向下圆整，即可得到一个位于0和数组长度减1之间的整数。

```
var rand1 = Math.floor(Math.random() * words1.length);
var rand2 = Math.floor(Math.random() * words2.length);
var rand3 = Math.floor(Math.random() * words3.length);
```

rand1将是一个位于0和数组words1的最后一个索引之间的整数。

rand2和rand3与此类似。

④ 将随机选择的单词拼接起来（并在单词之间添加空格，以提高可读性），生成时髦的广告语。

定义另一个用于存储广告语的变量。

将每个随机数用作包含单词的数组的索引。

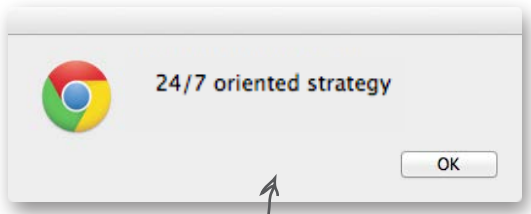
```
var phrase = words1[rand1] + " " + words2[rand2] + " " + words3[rand3];
```

⑤ 就要大功告成了：有了广告语，只需将其显示出来即可。为此，我们像往常那样使用`alert`。

```
alert(phrase);
```

⑥ 编写最后一行代码后，再看一眼所有的代码，享受享受成就感，再在浏览器中加载它们。来试驾一下，欣赏一下生成的广告语。

这是我们运行时生成的广告语！



只需不断重新加载网页，就能生成无穷无尽的创业公司广告语。（准确地说，并非无穷无尽的，但完全实现了让简单代码激动人心的目标！）

世上没有愚蠢的问题

问：数组中元素的排列顺序重要吗？

答：在大多数情况下都很重要，但也要看情况。在泡泡玩具反斗城公司的配方得分数组中，排列顺序很重要，因为该数组的索引指出了得分对应的配方，如0号配方的得分为60，该得分存储在索引0处。如果改变该数组中得分的排列顺序，将导致实验失败！然而，在有些情况下，排列顺序可能无关紧要。例如，使用数组来存储一系列随机选择的单词时，你不关心这些单词的排列顺序，因此它们在数组中按什么样的顺序排列都无所谓。但如果你要按字母顺序排列它们，那么排列顺序就很重要。因此，排列顺序是否重要完全取决于你要如何使用数组。你可能发现，使用数组时，排列顺序有关系比无关紧要的可能性更大。

问：在一个数组中，可存储多少个元素？

答：从理论上说，你想存储多少就能存储多少；但实际上，可存储的元素数受制于计算机的内存量。每个数组元素都要占据一定的内存空间。别忘了，JavaScript代码是在浏览器中运行的，而浏览器只是计算机运行的众多程序之一。如果你不断地在数组中添加元素，最终将耗尽内存空间。然而，可在数组中添加的最大元素数即使达不到数百万个，也可达到数千个，具体数量取决于元素的类型；这足以满足大多数情况下的需求。别忘了，数组包含的元素越多，程序的运行速度就越慢。因此在大多数情况下，都应将数组的长度限制在合理范围内，如数百个元素。

问：可以创建空数组吗？

答：可以。事实上，稍后你就将看到一个使用空数组的示例。要创建空数组，可这样编写代码：

```
var emptyArray = [ ];
```

如果数组开始是空的，可在以后添加元素。

问：到目前为止，数组元素要么是字符串，要么是数字；可将其他的值添加到数组中吗？

答：可以。事实上，在数组中几乎可存储任何JavaScript值，包括数字、字符串、布尔值、其他数组以及后面将介绍的对象。

问：一个数组中所有值的类型都必须相同吗？

答：不用，但我们通常在一个数组中存储相同类型的值。不同于众多其他的语言，JavaScript不要求一个数组中所有值的类型都相同。然而，如果在同一个数组中存储不同类型的值，使用这些值时就得特别小心。其原因如下：假设你有数组[1, 2, "fido", 4, 5]，并编写代码来检查其中的值是否大于2。当你检查"fido"是否大于2时结果将如何呢？为避免所做的事情不合理，在代码中使用该数组中的每个值之前，必须检查其类型。你当然能够对类型进行检查（这将在本书后面介绍），但一般而言，在数组中存储类型相同的值时，处理起来将更容易、更安全。

问：如果试图访问索引太大或太小（如小于0）的数组元素，结果将如何？

答：如果你有一个下面这样的数组，并试图访问a[10]或a[-1]，得到的结果将是undefined：

```
var a = [1, 2, 3];
```

因此，你要么确保访问数组元素时使用的索引有效，要么通过检查确认获得的值不是undefined。

问：我知道，可使用索引0来获取数组的第一个元素，但如何获取最后一个数组元素呢？我必须始终准确地知道数组包含多少个元素吗？

答：可利用属性length来获取最后一个数组元素。你知道，属性length总是比数组的最后一个索引大1，因此要获取最后一个数组元素，可编写这样的代码：

```
myArray[myArray.length - 1];
```

上述JavaScript代码获取数组的长度，将其减1，再获取该索引处的元素。因此，如果数组有10个元素，这行代码将获取索引9处的元素，这正是你希望的。每当你要获取数组的最后一个元素，又不知道数组到底包含多少个元素时，都可使用这种技巧。

回到泡泡玩具反斗城公司



嘿，很高兴大家来到这里。我们刚对大量新泡泡配方进行了测试，下面是所有这些配方的得分。我亟需你们帮助我理解这些数据。我期待你们编写代码，生成下面描述的报表。

泡泡玩具反斗城CEO

```
var scores = [60, 50, 60, 58, 54, 54,
              58, 50, 52, 54, 48, 69,
              34, 55, 51, 52, 44, 51,
              69, 64, 66, 55, 52, 61,
              46, 31, 57, 52, 44, 18,
              41, 53, 55, 61, 51, 44];
```

新泡泡配方的得分。

我们需要生成的报表。

泡泡玩具反斗城

我亟需一个下面这样的报表，让我能够迅速决定该采用哪种泡泡配方！你能编写生成这种报表的代码吗？

——泡泡玩具反斗城CEO

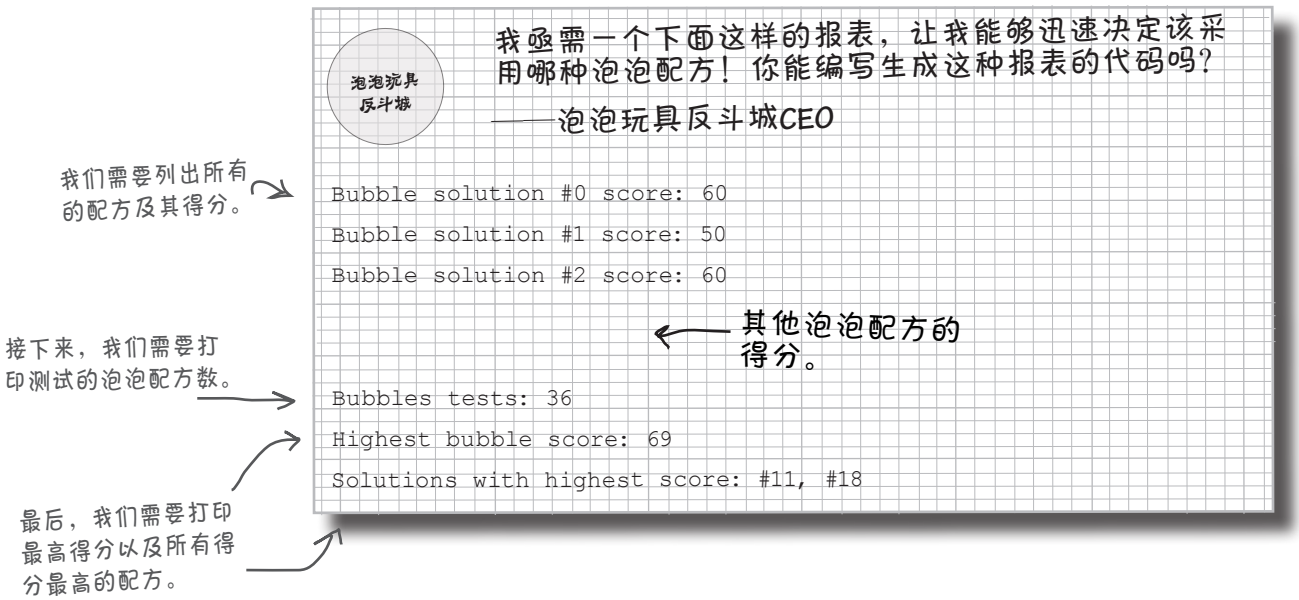
```
Bubble solution #0 score: 60
Bubble solution #1 score: 50
Bubble solution #2 score: 60
```

← 其他泡泡配方的得分。

```
Bubbles tests: 36
Highest bubble score: 69
Solutions with highest score: #11, #18
```


图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

咱们来详细研究该CEO需要的报表。



考考你的脑力

花点时间想想如何制作这个泡泡得分报表。考虑如何正确地输出报表的各项内容，并将你的想法记录在下面。

办公室商谈



Judy：我们首先要做的是显示所有的得分及其配方号。

Joe：配方号其实就是得分在数组中的索引，对吧？

Judy：完全正确。

Frank：让我想想。因此，我们需要获取每个得分，打印其索引（即配方号），再打印相应的得分。

Judy：你说的没错，而得分其实就是数组中相应的值。

Joe：因此，对于10号配方，其得分为`scores[10]`。

Judy：没错。

Frank：但得分很多，如何编写输出所有得分的代码呢？

Judy：使用迭代。

Frank：哦，你是说使用`while`循环什么的？

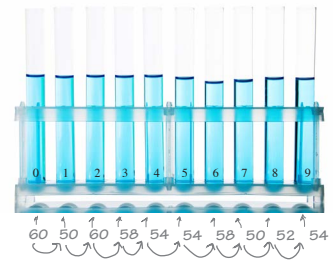
Judy：对，我们从零遍历到数组长度……哦，是数组长度减1，以遍历所有的值。

Joe：这听起来完全行得通。咱们来编写一些代码；我想我们已经知道该如何做了。

Judy：我也是这么认为的！咱们来编写代码吧，稍后再来商谈如何处理报表的其他部分。

如何迭代数组

你的目标是生成类似下面的输出：




```
Bubble solution #0 score: 60
Bubble solution #1 score: 50
Bubble solution #2 score: 60
.
.
.
Bubble solution #35 score: 44
```

这里是3~34号配方的得分。为节省纸张（如果你阅读的是电子版，就是节省比特），这里省略了它们。

为此，我们依次输出索引0、1、2等处的得分，直到到达数组的最后一个索引。你知道如何使用`while`循环，咱们来看看如何使用它来输出所有的得分：

稍后将介绍一种更好的方式。

```
var scores = [60, 50, 60, 58, 54, 54, 58, 50, 52, 54, 48, 69,
              34, 55, 51, 52, 44, 51, 69, 64, 66, 55, 52, 61,
              46, 31, 57, 52, 44, 18, 41, 53, 55, 61, 51, 44];

var output;
```

我们将在下面的循环中使用这个变量来存储要输出的字符串。

```
var i = 0;
```

创建一个用于存储当前索引的变量。

```
while (i < scores.length) {
  output = "Bubble solution #" + i + " score: " + scores[i];
  console.log(output);
  i = i + 1;
}
```

只要索引小于数组的长度，就继续循环。

创建一个作为一行输出的字符串，其中包含配方号（它刚好是数组索引）和得分。

使用*console.log*输出这个字符串。

最后，在再次循环前将索引加1。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权



代码冰箱贴

我们编写了一些代码，用于找出哪些口味的冰淇淋中有泡泡糖。我们使用冰箱贴将所有的代码都正确地贴到了冰箱上，但这些冰箱贴都脱落了。你的任务是将它们重新贴好。请注意，其中有一些冰箱贴是多余的。请查看本章末尾的答案，再继续往下阅读。

```
while (i < hasBubbleGum.length)
```

```
i = i + 2;
```

```
{
```

```
}
```

```
i = i + 1;
```

```
}
```

```
{
```

```
var i = 0;
```

```
{
```

```
if (hasBubbleGum[i])
```

```
while (i > hasBubbleGum.length)
```

```
var products = ["Choo Choo Chocolate",
                "Icy Mint", "Cake Batter",
                "Bubblegum"];
```

```
var hasBubbleGum = [false,
                    false,
                    false,
                    true];
```

```
console.log(products[i] +
    " contains bubble gum");
```

在这里重新排列冰箱贴。

这是我们期望的输出。

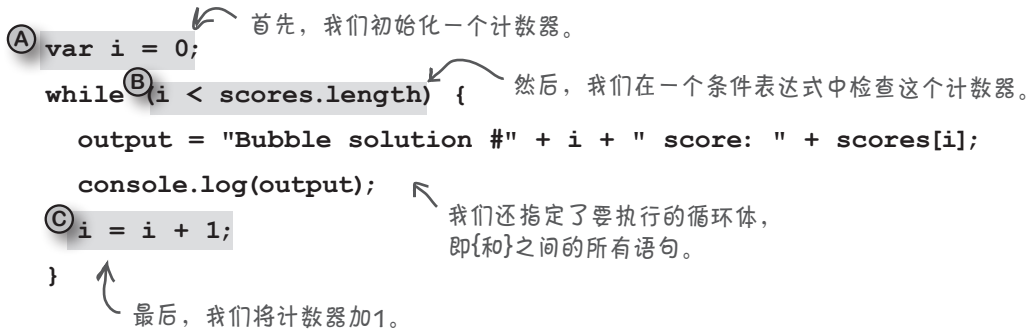
JavaScript控制台

```
Bubblegum contains bubble gum
```



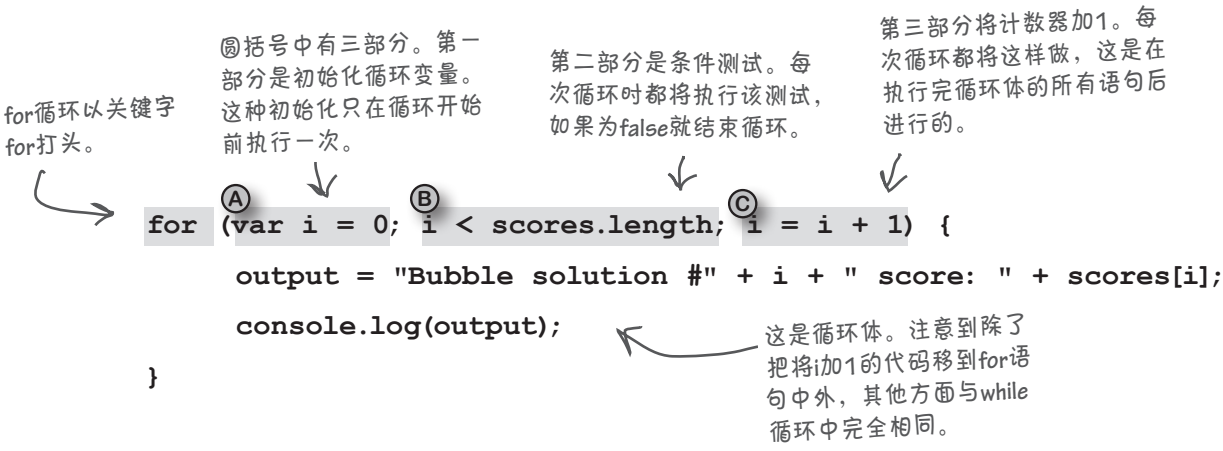
等等，还有一种更好的数组迭代方式

实在抱歉，都到第4章了，我们还没有介绍**for循环**，真不敢相信。你可将for循环视为while循环的表亲，它们的功能基本相同，只是for循环使用起来通常更方便些。来看刚才使用的while循环，我们将演示如何将其转换为for循环。




```
Ⓐ var i = 0;
while Ⓑ(i < scores.length) {
  output = "Bubble solution #" + i + " score: " + scores[i];
  console.log(output);
Ⓒ i = i + 1;
}
```

如果使用for循环，这些功能实现起来将容易得多。下面就来看看。




```
for (Ⓐvar i = 0; Ⓑi < scores.length; Ⓒi = i + 1) {
    output = "Bubble solution #" + i + " score: " + scores[i];
    console.log(output);
}
```



磨笔上阵

请重写两页前的冰箱贴代码，用for循环代替其中的while循环。如果你需要帮助，请参阅前一页中while循环的每个组成部分，看看它们分别对应于for循环的哪个部分。

```
var products = ["Choo Choo Chocolate",
                "Icy Mint", "Cake Batter",
                "Bubblegum"];

var hasBubbleGum = [false,
                    false,
                    false,
                    true];

var i = 0;
while (i < hasBubbleGum.length)
{
  if (hasBubbleGum[i])
  {
    console.log(products[i] +
        " contains bubble gum");
  }
  i = i + 1;
}
```

在这里编写代码。



这是网页的标准HTML内容。需要编写的HTML不多，只需为嵌入脚本做好准备工作即可。

```
<!doctype html>

<html lang="en">

<head>

  <meta charset="utf-8">

  <title>Bubble Factory Test Lab</title>

  <script>

    var scores = [60, 50, 60, 58, 54, 54,
                  58, 50, 52, 54, 48, 69,
                  34, 55, 51, 52, 44, 51,
                  69, 64, 66, 55, 52, 61,
                  46, 31, 57, 52, 44, 18,
                  41, 53, 55, 61, 51, 44];

    var output;

    for (var i = 0; i < scores.length; i = i + 1) {

        output = "Bubble solution #" + i +
                    " score: " + scores[i];

        console.log(output);

    }

  </script>

</head>

<body></body>

</html>
```

这是存储泡泡配方得分的数组。

这是迭代所有泡泡配方得分的for循环。

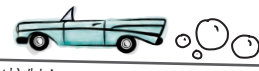
每次循环时都创建一个字符串，其中包含i的值（泡泡配方号）和scores[i]的值（泡泡配方的得分）。

接下来，我们将这个字符串输出到控制台。就这么简单！该尝试运行这些代码了！

（另外，注意到我们将这个字符串的内容放在了两行中。只要不在用于分隔字符串的配对引号之间换行，就没有问题。在这里，我们在一个拼接运算符（+）后面换行，因此没有问题。请一定完全按这里的显示输入。）

测试报表

将这个文件保存为bubbles.html，并在浏览器中加载它。确保控制台可见（如果你在加载这个网页后打开控制台，可能需要重新加载网页），欣赏欣赏你为泡泡玩具反斗城CEO制作的出色报表。



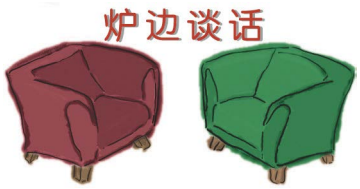
JavaScript控制台

```
Bubble solution #0 score: 60
Bubble solution #1 score: 50
Bubble solution #2 score: 60
Bubble solution #3 score: 58
Bubble solution #4 score: 54
Bubble solution #5 score: 54
Bubble solution #6 score: 58
Bubble solution #7 score: 50
Bubble solution #8 score: 52
Bubble solution #9 score: 54
Bubble solution #10 score: 48
Bubble solution #11 score: 69
Bubble solution #12 score: 34
Bubble solution #13 score: 55
Bubble solution #14 score: 51
Bubble solution #15 score: 52
Bubble solution #16 score: 44
Bubble solution #17 score: 51
Bubble solution #18 score: 69
Bubble solution #19 score: 64
Bubble solution #20 score: 66
Bubble solution #21 score: 55
Bubble solution #22 score: 52
Bubble solution #23 score: 61
Bubble solution #24 score: 46
Bubble solution #25 score: 31
Bubble solution #26 score: 57
Bubble solution #27 score: 52
Bubble solution #28 score: 44
Bubble solution #29 score: 18
Bubble solution #30 score: 41
Bubble solution #31 score: 53
Bubble solution #32 score: 55
Bubble solution #33 score: 61
Bubble solution #34 score: 51
Bubble solution #35 score: 44
```

与CEO要求的顺序完全相同。

能够在一个报表中看到所有泡泡配方的得分是挺好的，但要找出最高得分还是很难。我们需要处理其他报表需求，便于找出最佳配方。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权



今晚主题：while和for循环回答“谁更重要”的问题

while循环

开什么玩笑！我是JavaScript中最通用的循环结构。我不是非得使用傻乎乎的计数器，而是能使用任何类型的条件。本书先介绍的是我，还有人没有注意到这一点吗？

还有，你们注意到了吗？`for`循环没有一点幽默感。我是说，如果我整天做的都是无聊的迭代，我猜我也会变成那样。

是吗？我认为这不可能是真的。

本书前面都说过了，`for`和`while`循环的功能几乎相同，这怎么可能呢？

for循环

我不喜欢你说话的口气。

别贫了。可你注意到了吗，程序员十次有九次都会使用`for`循环。

另外，对于元素数固定的数组，使用`while`循环进行迭代是笨拙而糟糕的做法。

啊哈，你这是承认了我们之间的差距并没有那么大吗？

让我来告诉你原因。

while循环

根本没有你说得那么夸张。我见到的大多数迭代甚至不包含计数器，就像下面这样：

```
while (answer != "forty-two")
```

使用for循环试试！

啊，这样也行，真难以置信。

给猪涂上口红，它还是猪。

不但更擅长，代码也更漂亮。

for循环

使用while循环时，必须使用单独的语句来初始化和递增计数器。大刀阔斧地修改代码时，如果不小心移动或删除了这些语句，可能带来极其糟糕的后果。使用for循环时，这些代码都封装在for语句中，程序员看得清清楚楚，想不小心修改或删除它们都难。

没问题：

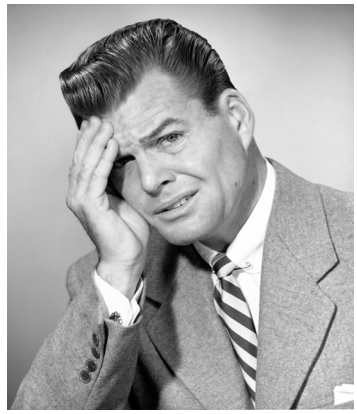
```
for (;answer != "forty-two";)
```

当然行。

你就会耍贫嘴！你还认为你更擅长处理一般性条件吗？

这又不是选美比赛。

别这么唠嗦好不好



你编写了大量像下面这样的代码：

假设myImportantCounter包含一个数字，如0。

将这个变量加1。

```
myImportantCounter = myImportantCounter + 1;
```

这条语句执行完毕后，myImportantCounter比原来大1。

事实上，鉴于这种语句很常见，JavaScript提供了一种快捷方式，称为后递增运算符（post-increment operator）。这个运算符的名字虽然奇特，但实际上非常简单。通过使用后递增运算符，可将上述代码替换为下面的代码：

只需在变量名后面加上++即可。

```
myImportantCounter++;
```

这条语句执行完毕后，myImportantCounter比原来大1。

当然，如果没有后递减运算符（post-decrement operator），会让人觉得不对头。你可将后递减运算符用于变量，将其值减1，如下所示：

只需在变量名后面加上--即可。

```
myImportantCounter--;
```

这条语句执行完毕后，myImportantCounter比原来小1。

为何在这里介绍后递增运算符呢？因为它常用于`for`语句中。下面来使用后递增运算符，让我们的代码更整洁些。

使用后递增运算符重新编写for循环

咱们来重新编写代码，并通过测试来确认其作用与以前一样：

```
var scores = [60, 50, 60, 58, 54, 54,
              58, 50, 52, 54, 48, 69,
              34, 55, 51, 52, 44, 51,
              69, 64, 66, 55, 52, 61,
              46, 31, 57, 52, 44, 18,
              41, 53, 55, 61, 51, 44];

for (var i = 0; i < scores.length; i++) {

    var output = "Bubble solution #" + i +
                 " score: " + scores[i];

    console.log(output);

}
```

我们只修改循环变量的递增方式，使用后递增运算符。

快速测试

来做一下快速测试，确认使用后递增运算符也可行。保存文件bubbles.html，并重新加载它。你看到的报表应该与以前一样。



JavaScript控制台

```
Bubble solution #0 score: 60
Bubble solution #1 score: 50
Bubble solution #2 score: 60
 ...
Bubble solution #34 score: 51
Bubble solution #35 score: 44
```

报表与以前完全一样。

为节省篇幅，这里没有显示所有配方的得分，但它们都显示到了控制台中。

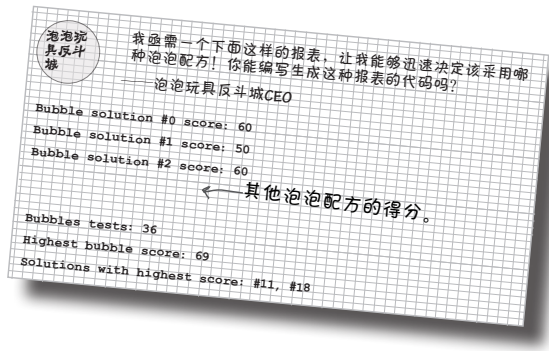
图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

办公室商谈（续）



Judy：是的，为此我们首先需要确定总共测试了多少个配方。这很简单，它就是数组scores的长度。

Joe：没错。我们还必须找出最高得分，还有获得最高得分的配方。



Judy：是的，但最后一项任务最棘手。咱们先来找出最高得分。

Joe：从这里着手看起来不错。

Judy：为此，需要创建一个变量，用于在遍历数组期间存储最高得分，下面我来编写一些伪代码。

```
声明变量highScore，并将其设置为零
FOR循环：i=0; i<scores.length; i++
    显示当前泡泡配方的得分scores[i]
    IF scores[i]>highScore
     将highscore设置为scores[i];
    IF语句到此结束
FOR循环到此结束
显示highScore
```

声明一个用于存储最高得分的变量。

在每次循环中，都检查当前配方的得分是否更高；如果是，就将其作为新的最高得分。

循环结束后，只需显示最高得分即可。

Joe：很好，只需添加几行代码就可完成这项任务。

Judy：遍历每个数组元素时，都检查它是否大于highScore；如果是，就将其作为新的最高得分。循环结束后，只需显示最高得分即可。



磨笔上阵

请补全下面的代码，以实现前一页中找出最高得分的伪代码；再据此更新bubbles.html中的代码，并尝试在浏览器中重新加载该网页。查看控制台中的结果，并据此填写下面的控制台显示结果中空白的内容：测试的泡泡配方数和最高得分。查看本章末尾的答案，再接着往下阅读。

```
var scores = [60, 50, 60, 58, 54, 54,
              58, 50, 52, 54, 48, 69,
              34, 55, 51, 52, 44, 51,
              69, 64, 66, 55, 52, 61,
              46, 31, 57, 52, 44, 18,
              41, 53, 55, 61, 51, 44];

var highScore = _____;
var output;
for (var i = 0; i < scores.length; i++) {
    output = "Bubble solution #" + i + " score: " + scores[i];
    console.log(output);
    if (_____________ > highScore) {
        _____________ = scores[i];
    }
}
console.log("Bubbles tests: " + _____________);
console.log("Highest bubble score: " + _____________);
```

请补全这里的代码。

请补全这里的控制台输出。

JavaScript控制台

```
Bubble solution #0 score: 60
Bubble solution #1 score: 50
Bubble solution #2 score: 60
 ...
Bubble solution #34 score: 51
Bubble solution #35 score: 44
Bubbles tests: _____
Highest bubble score: _____
```



就要大功告成了！余下的唯一工作是找出得分最高的配方并将其打印出来。别忘了，得分最高的配方可能有多个。

得分最高的配方可能有多个。需要存储多项内容时，该使用什么呢？当然是数组。是不是可以这样做：遍历数组scores，找出得分最高的配方，将它们加入一个数组，再在报表中显示该数组？绝对可以这样做，为此必须学习如何新建一个空数组以及如何在这个数组中添加新元素。

这是我们尚未完成的报表部分。

泡泡玩具反斗城

我亟需一个下面这样的报表，让我能够迅速决定该采用哪种泡泡配方！你能编写生成这种报表的代码吗？

——泡泡玩具反斗城CEO

```
Bubble solution #0 score: 60
Bubble solution #1 score: 50
Bubble solution #2 score: 60
```

←其他泡泡配方的得分。

```
Bubbles tests: 36
Highest bubble score: 69
Solutions with highest score: 11, 18
```

创建空数组并在其中添加元素



着手完成生成报表的代码前，先来感受一下如何创建空数组并在其中添加新元素。你已经知道了如何创建非空数组，如下所示：

```
var genres = ["80s", "90s", "Electronic", "Folk"];
```

这是一个数组字面量，我们详细地指定了该数组包含的元素。

你也可以省略初始值，从而创建一个空数组：

```
var genres = [];
```

一个可使用的新数组，它不包含任何元素，因此长度为零。

这也是一个数组字面量，但不包含任何元素。

你已经知道如何给数组添加新元素。只需给指定索引处的元素赋值即可，如下所示：

```
var genres = [];
genres[0] = "Rockabilly";
genres[1] = "Ambient";
var size = genres.length;
```

新建一个数组元素，它存储的是字符串“Rockabilly”。

新建另一个数组元素，它存储的是字符串“Ambient”。

变量size的值为2，即数组genres的长度。

添加新元素时，必须小心地指定索引，否则数组将是稀疏的（sparse），即存在“空洞”（如索引0和2处有值，而索引1处没有值）。数组稀疏未必是坏事，但使用时必须特别小心。还有一种添加新元素的方式，就是使用push方法。使用这种方式时无需指定索引，其工作原理如下：

```
var genres = [];
genres.push("Rockabilly");
genres.push("Ambient");
var size = genres.length;
```

在下一个可用的索引（这里为0）处新建一个元素，并将其值设置为“Rockabilly”。

在下一个可用的索引（这里为1）处新建另一个元素，并将其值设置为“Ambient”。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

世上没有愚蠢的问题

问：你说过应将变量声明放在开头，可for语句的第一部分包含变量声明和初始化，这是怎么回事呢？

答：是的，将变量声明放在开头（对于全局变量，是文件开头；对于局部变量，是函数开头）是一种不错的做法，但有时候在要用时再声明变量更合理，而for语句就属于这样的情况。通常，循环变量（如i）仅用于迭代，循环结束后就不再需要它了。当然，循环结束后，你也可能需要使用变量i，但通常不是这样的，因此在for语句中声明它能让代码更整洁。

问：语法myarray.push(value)到底是什么意思呢？

答：有一点我们一直没有说，那就是在JavaScript中，数组实际上是一种特殊的对象，而对象可以有相关联的函数，用于操作对象。这将在下一章介绍。因此，你可将push视为一个可对myarray进行操作的函数。在这里，函数push在数组中添加一个元素，而元素本身是以实参的方式传递给push的。因此，代码genres.push("Metal");调用函数push，并将字符串实参"Metal"传递给它。函数push将这个实参作为新元素添加到数组genres末尾。因此，看到myarray.push(value)时，你可以这样想：这是要在数组myarray末尾添加一个新元素。

问：能更详细地说说稀疏数组吗？

答：稀疏数组是有些索引处有值，而其他索引处没有值的数组。创建稀疏数组易如反掌，如下所示：

```
var sparseArray = [ ];
sparseArray[0] = true;
spraseArray[100] = true;
```

在这个示例中，sparseArray只有两个值，这两个值都是true，分别位于索引0和100处。其他索引处的值都是undefined。这个数组虽然只包含两个值，但其长度却是101。

问：假设有一个长度为10的数组，在索引10 000处添加一个新元素后，索引10～9999处的值将是什么？

答：这些数组索引处的值都为undefined。你可能还记得，未初始化的变量的值为undefined。因此，你可以这样认为：这犹如创建了9989个变量，但没有对它们进行初始化。别忘了，这些变量虽然没有值，但它们都要占用计算机内存，因此除非有充分的理由，否则千万不要创建稀疏数组。

问：迭代数组时，如果其中有些元素的值为undefined，使用元素前必须确认它不是undefined吗？

答：如果你认为数组可能是稀疏的，使用其中的元素前必须确认其值不是undefined，那怕该数组只有一个元素的值是undefined。将元素的值显示到控制台时，这不是什么大问题，但你通常要以某种方式使用元素的值，如使用它来执行计算。在这种情况下，使用undefined即使不会导致错误，也会导致意外的行为。要检查元素的值是不是undefined，可编写类似于下面的代码：

```
if (myarray[i] == undefined) {
  ...
}
```

请注意，undefined是一个值，而不是字符串，因此不能在前后加上引号。

问：本书前面创建的数组都是字面量，还有其他创建数组的方式吗？

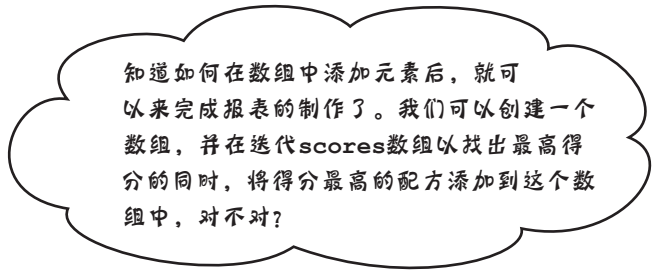
答：有。你可能见过这样的语法：

```
var myarray = new Array(3);
```

它新建一个数组，其中包含3个空位置，即长度为3，但不包含任何值。然后，你可以像通常那样填充这些空位置：在数组myarray的索引0、1和2处填充值。在此之前，myarray中元素的值都是undefined。

以此种方式创建的数组与数组字面量没什么两样。实际上，你更常用的是字面量语法，本书后面也通常使用这种语法。

对于上述语法，现在暂时不要考虑其中的细节，如new以及Array的首字母为何大写；本书后面将详细介绍。





Judy：对。我们首先创建一个空数组，用于存储得分最高的配方，并在迭代数组scores的过程中，将得分最高的配方添加到这个数组中。

Frank：很好，咱们开始吧。

Judy：等等，我认为我们可能需要使用一个单独的循环。

Frank：是吗？好像应该有办法在既有循环中完成这项任务。

Judy：是的，必须使用一个单独的循环。原因如下：要找出得分最高的所有配方，必须知道最高得分是多少。因此我们需要两个循环，一个找出最高得分（这已经编写好了），另一个找出获得最高得分的所有配方。

Frank：哦，我明白了。在第二个循环中，我们把每个配方的得分与最高得分进行比较；如果相同，就将配方的索引添加到用于存储得分最高配方的数组中。

Judy：完全正确！咱们来编写代码吧。



磨笔上阵

你能编写一个循环，找出得分最高的所有配方吗？请在下面试一试，再查看下一页的答案。并对其进行测试。

别忘了，变量highScore存储了最高得分，你可在下面的代码中使用这个变量。

我们将使用这个新数组来存储得分最高的配方。

```
var bestSolutions = [];
for (var i = 0; i < scores.length; i++) {

}
```

在这里编写代码。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权



你能编写一个循环，找出得分最高的所有配方吗？答案如下。

首先，新建一个数组，用于存储得分最高的所有配方。

接下来，迭代整个scores数组，找出得分最高的元素。

```
var bestSolutions = [];

for (var i = 0; i < scores.length; i++) {
    if (scores[i] == highScore) {

        bestSolutions.push(i);

    }
}

console.log("Solutions with the highest score: " + bestSolutions);
```

每次循环时，都将索引i处的得分与highScore进行比较；如果它们相等，就使用函数push将该索引添加到数组bestSolutions中。

最后，可以显示得分最高的泡泡配方了。注意到我们使用了console.log来显示数组bestSolutions。我们原本可以再创建一个循环，逐个显示这个数组的元素，但console.log能够替我们完成这样的任务。（如果你查看输出，将发现它还在数组元素之间添加了逗号！）



请研究上面的代码。如果你一觉醒来后，函数push消失了，该如何办呢？你能不再使用push重写上面的代码吗？请在下面写出这样的代码。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

测试最终的报表



将显示得分最高泡泡配方代码加入bubbles.html，并再次对该网页进行测试。下面显示了完整的JavaScript代码：

```
var scores = [60, 50, 60, 58, 54, 54,
              58, 50, 52, 54, 48, 69,
              34, 55, 51, 52, 44, 51,
              69, 64, 66, 55, 52, 61,
              46, 31, 57, 52, 44, 18,
              41, 53, 55, 61, 51, 44];

var highScore = 0;
var output;
for (var i = 0; i < scores.length; i++) {
    output = "Bubble solution #" + i + " score: " + scores[i];
    console.log(output);
    if (scores[i] > highScore) {
        highScore = scores[i];
    }
}
console.log("Bubbles tests: " + scores.length);
console.log("Highest bubble score: " + highScore);

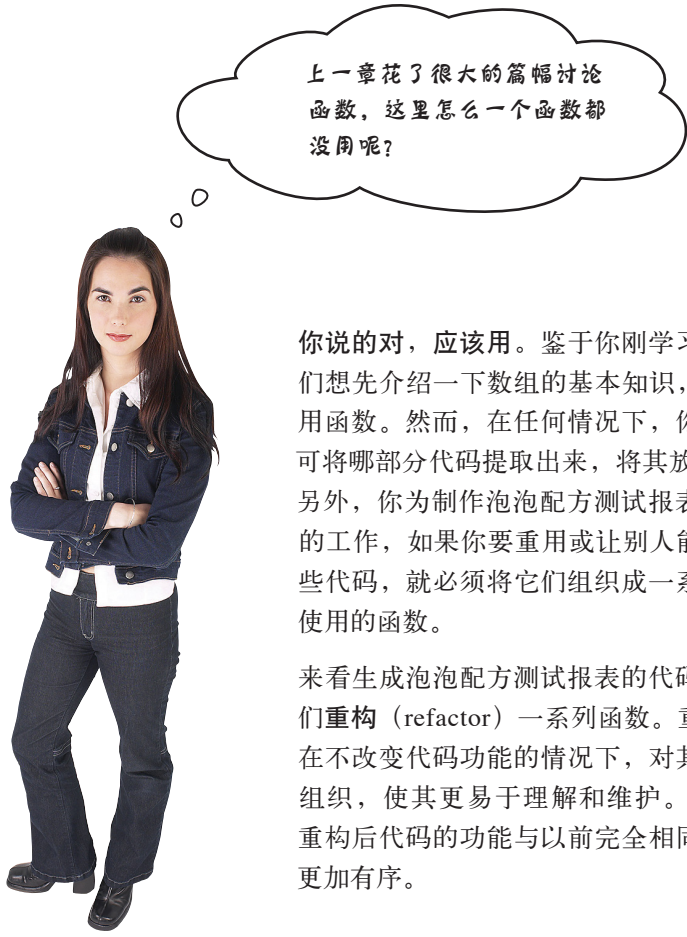
var bestSolutions = [];
for (var i = 0; i < scores.length; i++) {
    if (scores[i] == highScore) {
        bestSolutions.push(i);
    }
}
console.log("Solutions with the highest score: " + bestSolutions);
```

胜出的配方是……

11号和18号配方的得分最高，都是69！因此，在测试的这批配方中，它们是最好的。

JavaScript控制台

```
Bubble solution #0 score: 60
Bubble solution #1 score: 50
 ...
Bubble solution #34 score: 51
Bubbles tests: 36
Highest bubble score: 69
Solutions with the highest score: 11,18
```



你说的对，应该用。鉴于你刚学习函数，我们想先介绍一下数组的基本知识，再着手使用函数。然而，在任何情况下，你都应考虑可将哪部分代码提取出来，将其放到函数中。另外，你为制作泡泡配方测试报表做了大量的工作，如果你要重用或让别人能够重用这些代码，就必须将它们组织成一系列可直接使用的函数。

来看生成泡泡配方测试报表的代码，并将它们**重构**（refactor）一系列函数。重构指的是在不改变代码功能的情况下，对其进行重新组织，使其更易于理解和维护。换句话说，重构后代码的功能与以前完全相同，但组织更加有序。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

快速审视代码

下面来研究一下已编写的代码，确定要将哪些代码提取到函数中：



为泡泡玩具反斗城公司编写的代码。

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Bubble Factory Test Lab</title>
  <script>
    var scores = [60, 50, 60, 58, 54, 54,
                  58, 50, 52, 54, 48, 69,
                  34, 55, 51, 52, 44, 51,
                  69, 64, 66, 55, 52, 61,
                  46, 31, 57, 52, 44, 18,
                  41, 53, 55, 61, 51, 44];

    var highScore = 0;
    var output;

    for (var i = 0; i < scores.length; i++) {
      output = "Bubble solution #" + i + " score: " + scores[i];
      console.log(output);
      if (scores[i] > highScore) {
        highScore = scores[i];
      }
    }
    console.log("Bubbles tests: " + scores.length);
    console.log("Highest bubble score: " + highScore);

    var bestSolutions = [];

    for (var i = 0; i < scores.length; i++) {
      if (scores[i] == highScore) {
        bestSolutions.push(i);
      }
    }

    console.log("Solutions with the highest score: " + bestSolutions);
  </script>
</head>
<body> </body>
</html>
```

我们不想在处理得分的函数中声明数组scores，因为每次调用这个函数时，要处理的得分都不同。相反，我们将得分scores作为实参传递给这个函数，这样这个函数就能根据任何scores数组来生成结果。

这部分代码用来输出数组中的每个得分，并找出其中的最高得分。可将它们放在函数printAndGetHighScore中。

这部分代码根据最高得分找出最佳配方。可将它们放在函数getBestResults中。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

编写函数printAndGetHighScore

我们找出了要放在函数printAndGetHighScore中的代码。这些代码都是现成的，但要将其转换为函数，需要确定要给函数传递哪些实参以及它是否要返回值。

传入scores数组看起来是个不错的主意，因为这样可将这个函数用于处理其他的泡泡配方测试得分数组。另外，我们要返回这个函数找出的最高得分，让调用它的代码能够据此做些有趣的事情（我们将据此找出最佳配方）。

哦，还有一点：你通常希望函数**只做一件事**，并把这件事做好。然而，这个函数做了两件事：显示数组中的所有得分并找出最高得分。你可能想将它分成两个函数，但考虑到它做的两件事情都非常简单，我们就不这样做了。在实际工作中，可能需要将这个函数分成两个：printScores和getHighScore。但就这里而言，我们不打算这样做。重构后的代码如下：

我们创建了一个函数，它接受一个实参——数组*scores*。

```
function printAndGetHighScore(scores) {
    var highScore = 0;
    var output;
    for (var i = 0; i < scores.length; i++) {
        output = "Bubble solution #" + i + " score: " + scores[i];
        console.log(output);
        if (scores[i] > highScore) {
            highScore = scores[i];
        }
    }
    return highScore;
}
```

这些代码与以前完全相同，准确地说是看起来与以前完全相同，但使用的是形参*scores*，而不是全局变量*scores*。

我们新增了一行代码，它将*highScore*返回给调用函数的代码。

使用printAndGetHighScore重构代码

现在需要修改其他代码，以使用这个新函数。为此，只需调用新函数printAndGet-HighScore，并将变量highScore设置为其结果：

```
<!doctype html>
<html lang="en">
<head>
  <title>Bubble Factory Test Lab</title>
  <meta charset="utf-8">
  <script>
    var scores = [60, 50, 60, 58, 54, 54, 58, 50, 52, 54, 48, 69,
                  34, 55, 51, 52, 44, 51, 69, 64, 66, 55, 52, 61,
                  46, 31, 57, 52, 44, 18, 41, 53, 55, 61, 51, 44];

    function printAndGetHighScore(scores) {
        var highScore = 0;
        var output;
        for (var i = 0; i < scores.length; i++) {
            output = "Bubble solution #" + i + " score: " + scores[i];
            console.log(output);
            if (scores[i] > highScore) {
                highScore = scores[i];
            }
        }
        return highScore;
    }

    var highScore = printAndGetHighScore(scores);
    console.log("Bubbles tests: " + scores.length);
    console.log("Highest bubble score: " + highScore);

    var bestSolutions = [];

    for (var i = 0; i < scores.length; i++) {
      if (scores[i] == highScore) {
        bestSolutions.push(i);
      }
    }

    console.log("Solutions with the highest score: " + bestSolutions);
  </script>
</head>
<body> </body>
</html>
```

准备就绪的新函数。

现在只需调用这个函数，并传入数组scores，再将它返回的值赋给变量highScore。

接下来需要将这些代码重构为一个函数，并相应地修改其他代码。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权



磨笔上阵

让我们一起来完成这项重构任务。这里的目标是编写一个函数，它创建一个数组，其中包含得分最高的泡泡配方（可能有多个这样的配方，这就是我们使用数组来存储配方的原因所在）。我们将给这个函数传递数组scores，还有使用printAndGetHighScore计算得到的highScore。请补全下面的代码。答案见下一页，可不要偷看！请先尝试补全这些代码，这样你才能真正明白它们。

这是原来的代码；这里列出它们旨在供你参考。

```
var bestSolutions = [];
for (var i = 0; i < scores.length; i++) {
    if (scores[i] == highScore) {
        bestSolutions.push(i);
    }
}
console.log("Solutions with the highest score: " + bestSolutions);
```

我们完成了大致的轮廓，需要你来帮助补全！

```
function getBestResults(_____________, _______________) {
    var bestSolutions = _______;
    for (var i = 0; i < scores.length; i++) {
        if (_____________ == highScore) {
            bestSolutions._____________;
        }
    }
    return ___________________;
}

var bestSolutions = _________________(scores, highScore);
console.log("Solutions with the highest score: " + bestSolutions);
```

整合起来

重构好代码后，像下面这样修改bubbles.html，然后在浏览器中重新加载它。结果应与以前完全相同，但代码更加组织有序，可重用性更高。请创建自己的得分数组，并尝试重用这些代码！

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Bubble Factory Test Lab</title>
  <script>
    var scores = [60, 50, 60, 58, 54, 54, 58, 50, 52, 54, 48, 69,
                  34, 55, 51, 52, 44, 51, 69, 64, 66, 55, 52, 61,
                  46, 31, 57, 52, 44, 18, 41, 53, 55, 61, 51, 44];

    function printAndGetHighScore(scores) {
        var highScore = 0;
        var output;
        for (var i = 0; i < scores.length; i++) {
            output = "Bubble solution #" + i + " score: " + scores[i];
            console.log(output);
            if (scores[i] > highScore) {
                highScore = scores[i];
            }
        }
        return highScore;
    }

    function getBestResults(scores, highScore) {
        var bestSolutions = [];
        for (var i = 0; i < scores.length; i++) {
            if (scores[i] == highScore) {
                bestSolutions.push(i);
            }
        }
        return bestSolutions;
    }

    var highScore = printAndGetHighScore(scores);
    console.log("Bubbles tests: " + scores.length);
    console.log("Highest bubble score: " + highScore);

    var bestSolutions = getBestResults(scores, highScore);
    console.log("Solutions with the highest score: " + bestSolutions);

  </script>
</head>
<body> </body>
</html>
```

这是新函数getBestResults。

根据函数getBestResults返回的结果在报表中显示最佳配方。

做得好！还有一项任务：你能找出性价比最高的配方吗？有了这项数据，我们一定能占据整个泡泡配方市场。下面的数组包含每种配方的成本，你可据此找出性价比最高的配方。



这是成本数组，它包含得分数组中每个配方的成本。

```
var costs = [.25, .27, .25, .25, .25, .25,
             .33, .31, .25, .29, .27, .22,
             .31, .25, .25, .33, .21, .25,
             .25, .25, .28, .25, .24, .22,
             .20, .25, .30, .25, .24, .25,
             .25, .25, .27, .25, .26, .29];
```

这里的任务是什么呢？在最佳的泡泡配方（即得分最高的配方）中，找出成本最低的配方。好在我们有一个与scores数组对应的costs数组：在数组scores中，索引0处得分对应的泡泡配方的成本存储在数组costs的索引0处（为0.25）；在数组scores中，索引1处得分对应的泡泡配方的成本存储在数组costs的索引1处（为0.27），依此类推。也就是说，对于任何得分，都可在数组costs的相同索引处找到对应配方的成本。这样的数组也叫**平行数组**。

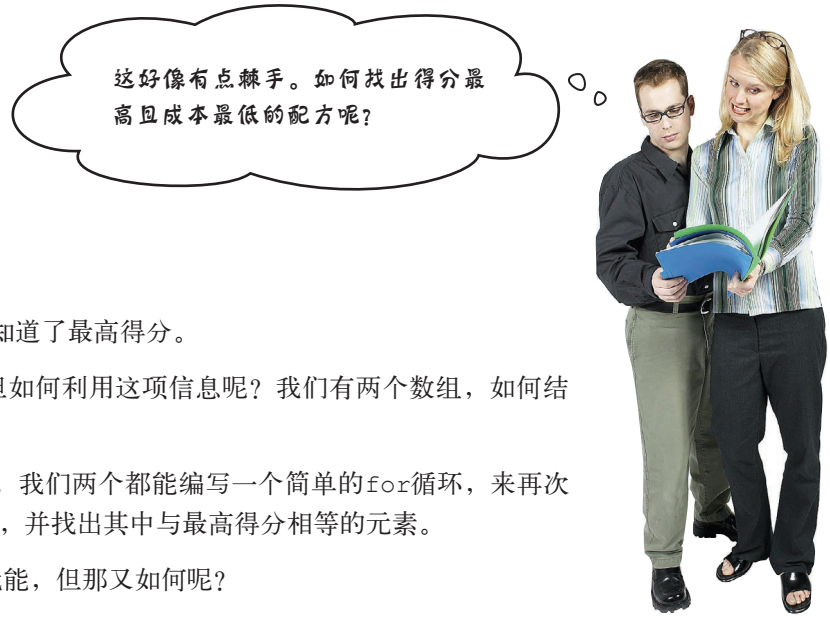
数组scores和costs是平行数组，因为每个得分和对应的成本都位于相同的索引处。

```
var costs = [.25, .27, .25, .25, .25, .25, .33, .31, .25, .29, .27, .22, ..., .29];
var scores = [60,  50,  60,  58,  54,  54,  58,  50,  52,  54,  48,  69,  ..., 44];
```

在数组costs中，索引0处的成本是0号泡泡配方的成本。

这两个数组中的其他成本和得分值也存在这种对应关系。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权



Judy：我们已经知道了最高得分。

Frank：是的，但如何利用这项信息呢？我们有两个数组，如何结合使用它们呢？

Judy：我敢肯定，我们两个都能编写一个简单的`for`循环，来再次遍历数组`scores`，并找出其中与最高得分相等的元素。

Frank：是的，我能，但那又如何呢？

Judy：每次遇到与最高得分相等的元素时，我们都需要检查其成本是否是最低的。

Frank：哦，我明白了，我们将声明一个变量，用于记录得分最高且成本最低的配方的索引。这好像有点绕口。

Judy：太对了。遍历完整个数组后，这个变量存储的索引就是得分最高且成本最低的配方的索引。

Frank：如果两个配方的成本相等，该如何办呢？

Judy：我们必须决定如何处理这种情形。要我说，哪个配方在前面就选哪个。当然，我们可以做更复杂的处理，但只要CEO同意，我们就采取这种处理方法吧。

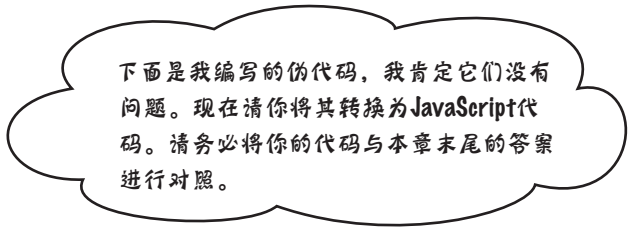
Frank：这已经很复杂了，我想我得先编写伪代码，再着手编写实际代码。

Judy：我也是这么认为的，管理多个数组的索引时，情况都可能比较棘手。咱们就来编写伪代码吧；从长远看，这样做是磨刀不误砍柴工。

Frank：好的，我来试一试。



练习





```
函数GETMOSTCOSTEFFECTIVESOLUTION (SCORE, COSTS, HIGHSCORE)
    声明变量cost并将其设置为100
    声明变量index
    FOR循环：var i=0; i < scores.length; i++
        IF scores[i]对应的泡泡配方的得分是最高的
            IF 变量cost的值大于该泡泡配方的成本
            THEN
                将变量index设置为i的值
                将变量cost设置为该泡泡配方的成本
            IF语句到此结束
        IF语句到此结束
    FOR循环到此结束
    返回index
```

```
function getMostCostEffectiveSolution(scores, costs, highscore) {
```

将上述伪代码转换为JavaScript代码，并写在这里。

```
}
var mostCostEffective = getMostCostEffectiveSolution(scores, costs, highScore);
console.log("Bubble Solution #" + mostCostEffective + " is the most cost effective");
```

选中的配方：11号配方

你刚编写的代码可帮助确定最终胜出的配方，即能够以最低的成本产生最多泡泡的配方。祝贺你通过处理大量的数据，给泡泡玩具反斗城公司提供了可用于作出商业决策的依据。

如果你像我们一样，一定很想知道11号配方都包含哪些东西。不用舍近求远，泡泡玩具反斗城公司的CEO说了，考虑到你没有收取任何编码费用，他愿意将这个配方告诉你。

下面就是11号配方的详细情况。请按这个配方制作，去室外吹些泡泡，再接着阅读下一章。阅读下一章前，别忘了还有要点和填字游戏！



11号泡泡配方

2/3杯洗洁精

1加仑水

两三调羹甘油（在药店和化学品商店都能买到）

说明：将这些成分放在一个大碗中并搅匀。

祝你玩得愉快！

在家里试试吧！







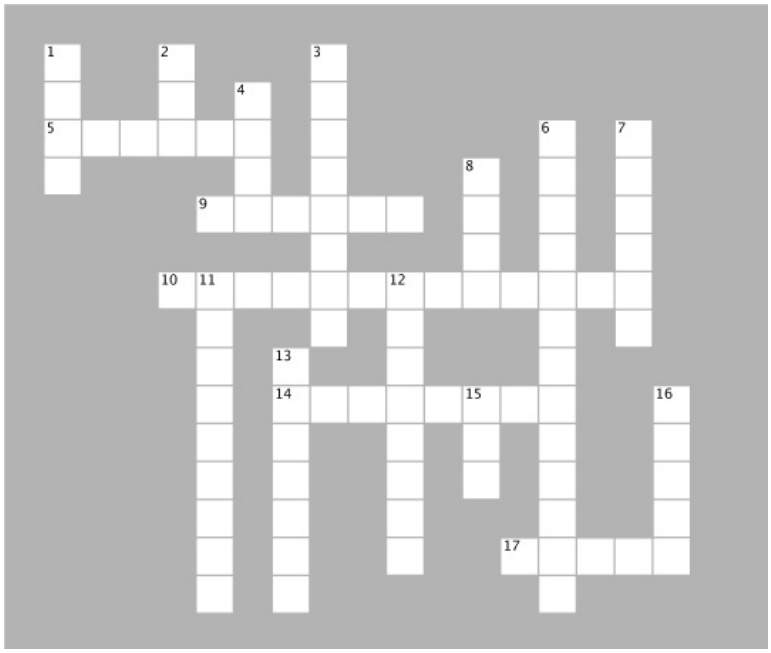
要点

- 数组是一种按顺序存储数据的**数据结构**。
- 数组包含一系列元素，其中每个元素都有**索引**。
- 数组索引从零开始，其中第一个元素的索引为零。
- 所有数组都有`length`属性，它指出了数组包含多少个元素。
- 要访问数组元素，可使用其索引。例如，`myArray[1]`访问元素1（数组中的第二个元素）。
- 试图访问不存在的元素将返回`undefined`。
- 给既有元素赋值将修改该元素的值。
- 给不存在的元素赋值将在数组中新建一个元素。
- 数组元素可以是任何类型的值。
- 在同一个数组中，不要求所有值的类型都相同。
- 要创建一个新数组，可使用**数组字面量表示法**。
- 要创建一个空数组，可使用语法`var myArray = [];`。
- 常使用`for`循环来迭代数组。
- `for`循环在一条语句中包含变量初始化、条件测试和变量递增。
- `while`循环最常用于不知道需要循环多少次时，它循环到条件满足为止。`for`循环最常用于知道循环需要执行多少次时。
- 稀疏数组指的是中间有值为`undefined`的元素的数组。
- 要将变量加1，可使用**后递增**运算符`++`。
- 要将变量减1，可使用**后递减**运算符`--`。
- 要在数组中添加新元素，可使用函数`push`。



JavaScript填字游戏

请完成这个填字游戏，让数组深深地刻在你的脑海中。



横向

5. 包含值为undefined的元素的数组。
9. 修改数组元素的值时，对元素执行的操作。
10. 认为其泡泡配方将胜出的人。
14. 重新组织代码，使其更容易理解和维护。
17. 数组的每个值都存储在一个什么处？

纵向

1. 用于在既有数组末尾添加新元素的函数。
2. 迭代数组时通常使用的一种循环。
3. 数组擅长存储一个还是多个值？
4. 数组的最后一个索引总是比数组长度小1还是大1？
6. 用于将循环变量加1的运算符。
7. 迭代数组时，通常根据哪个属性来确定什么时候该停止迭代？
8. 第一个数组元素的索引。
11. 为指定的数组元素的值。
12. 函数可帮助_____代码。
13. 数组是一种存储什么数据的数据结构？
15. 在本章中，得分最高的泡泡配方有多少个？
16. 要访问数组元素，可使用以方括号括起的什么？



磨笔上阵答案

下面的products数组存储了Jenn & Berry销售的各种冰淇淋口味。这些冰淇淋口味是按制作顺序添加到数组中的，请补全下面的代码，找出最后制作的冰淇淋口味。答案如下。

```
var products = ["Choo Choo Chocolate", "Icy Mint", "Cake Batter", "Bubblegum"];

var last = products.length - 1;

var recent = products[last];
```

要获取最后一个元素的索引，可将数组的长度减1。数组长度为4，因此最后一个元素的索引为3，这是因为索引是从0开始的。





代码冰箱贴答案

我们编写了一些代码，用于找出哪些口味的冰淇淋中有泡泡糖。我们使用冰箱贴将所有的代码都正确地贴到了冰箱上，但这些冰箱贴都脱落了。你的任务是将它们重新贴好。请注意，其中有一些冰箱贴是多余的。答案如下。

```
var products = ["Choo Choo Chocolate",
                "Icy Mint", "Cake Batter",
                "Bubblegum"];

var hasBubbleGum = [false,
                    false,
                    false,
                    true];

var i = 0;

while (i < hasBubbleGum.length)

{

if (hasBubbleGum[i])

{

console.log(products[i] +
    " contains bubble gum");

}

i = i + 1;

}
```

重新排列的冰箱贴。

多出来的冰箱贴。

```
{

i = i + 2;

while (i > hasBubbleGum.length)
```

这是我们期望的输出。

JavaScript控制台

```
Bubblegum contains bubble gum!
```





请重写两页前的冰箱贴代码，用for循环代替其中的while循环。如果你需要帮助，请参阅前一页中while循环的每个组成部分，看看它们分别对应于for循环的哪个部分。答案如下。

```
var products = ["Choo Choo Chocolate",
                "Icy Mint", "Cake Batter",
                "Bubblegum"];

var hasBubbleGum = [false,
                    false,
                    false,
                    true];

var i = 0;
while (i < hasBubbleGum.length)
{
    if (hasBubbleGum[i])
    {
        console.log(products[i] +
            " contains bubble gum");
    }
    i = i + 1;
}
```

在这里编写你的代码。

```
var products = ["Choo Choo Chocolate",
                "Icy Mint", "Cake Batter",
                "Bubblegum"];
var hasBubbleGum = [false,
                    false,
                    false,
                    true];
for (var i = 0; i < hasBubbleGum.length; i = i + 1) {
    if (hasBubbleGum[i]) {
        console.log(products[i] + " contains bubble gum");
    }
}
```



请补全下面的代码，以实现前一页中找出最高得分的伪代码；再据此更新bubbles.html中的代码，并尝试在浏览器中重新加载该网页。查看控制台中的结果，并据此填写下面的控制台显示结果中空白的内容：测试的泡泡配方数和最高得分。答案如下。

```
var scores = [60, 50, 60, 58, 54, 54,
              58, 50, 52, 54, 48, 69,
              34, 55, 51, 52, 44, 51,
              69, 64, 66, 55, 52, 61,
              46, 31, 57, 52, 44, 18,
              41, 53, 55, 61, 51, 44];

var highScore = 0;
var output;
for (var i = 0; i < scores.length; i++) {
    output = "Bubble solution #" + i + " score: " + scores[i];
    console.log(output);
    if (scores[i] > highScore) {
        highScore = scores[i];
    }
}
console.log("Bubbles tests: " + scores.length);
console.log("Highest bubble score: " + highScore);
```

请补全这里的代码。

请补全这里的控制台输出。

JavaScript控制台

```
Bubble solution #0 score: 60
Bubble solution #1 score: 50
Bubble solution #2 score: 60
 ...
Bubble solution #34 score: 51
Bubble solution #35 score: 44
Bubbles tests: 36
Highest bubble score: 69
```



下面是函数getMostCostEffectiveSolution的代码，它接受一个得分数组、一个成本数组和一个最高得分，并找出得分最高且成本最低的配方的索引。请将这些代码加入bubbles.html中，测试该网页，并确认你的结果与这里显示的相同。

函数getMostCostEffectiveSolution接受一个得分数组、一个成本数组和一个最高得分。

```
function getMostCostEffectiveSolution(scores, costs, highscore) {
    var cost = 100;
    var index;

    for (var i = 0; i < scores.length; i++) {
        if (scores[i] == highscore) {
            if (cost > costs[i]) {
                index = i;
                cost = costs[i];
            }
        }
    }
    return index;
}
var mostCostEffective = getMostCostEffectiveSolution(scores, costs, highScore);
console.log("Bubble Solution #" + mostCostEffective + " is the most cost effective");
```

我们将使用变量cost来记录最低成本。

我们将变量cost初始化为较大的数字；每当得分最高的配方有更低的成本时，都修改这个变量的值。

使用变量index来存储成本最低的配方的索引。

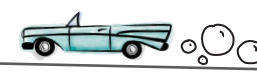
像以前那样迭代数组scores。

检查配方的得分是否最高。

如果是，就检查配方的成本。如果变量cost的值大于当前配方的成本，就说明找到了成本更低的配方，因此将该配方（它在数组中的索引）记录下来，并将其成本作为目前遇到的最低成本存储到变量cost中。

循环结束后，变量index存储的便是成本最低的配方的索引，因此我们将它返回给调用这个函数的代码。

接下来，将索引（泡泡配方号）显示到控制台。



JavaScript控制台

```
Bubble solution #0 score: 60
Bubble solution #1 score: 50
Bubble solution #2 score: 60
 ...
Bubble solution #34 score: 51
Bubble solution #35 score: 44
Bubbles tests: 36
Highest bubble score: 69
Solutions with the highest score: 11,18
Bubble Solution #11 is the most cost effective
```

最终的报告指出，11号泡泡配方在测试中胜出了，因为它以最低的成本产生了最多的泡泡。

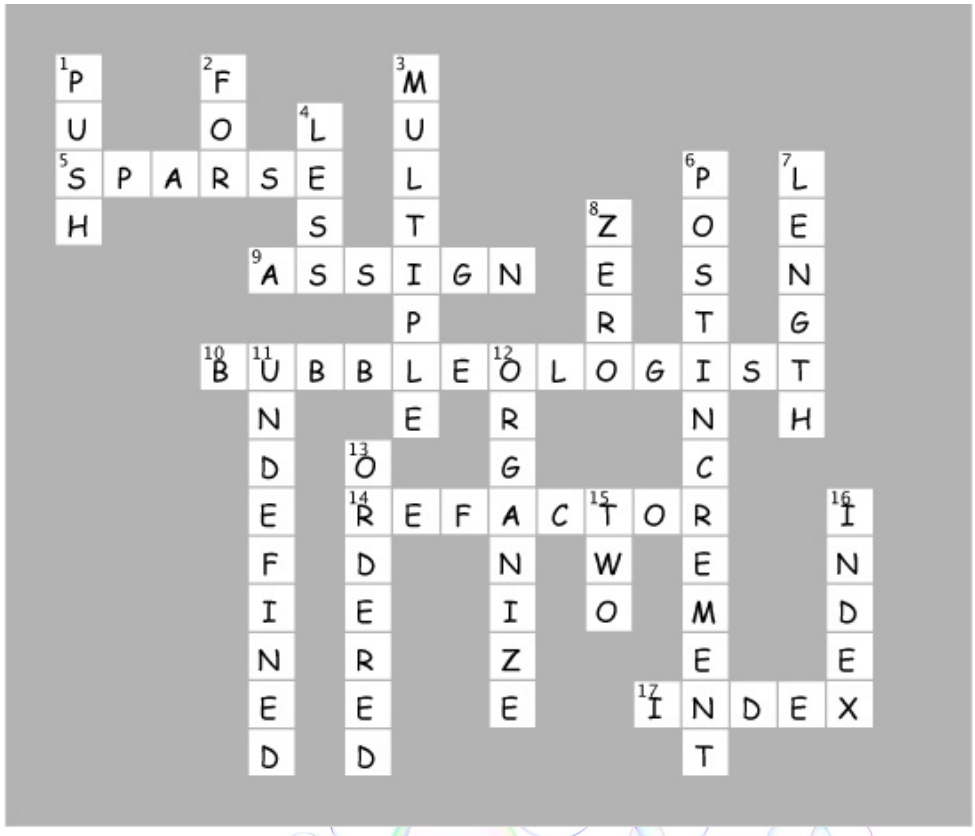
提示：也可使用数组bestSolutions来实现这种功能，这样就无需再次迭代所有的得分了。别忘了，数组bestSolutions存储了得分最高的配方的索引，因此在代码中，可以将数组bestSolutions的元素用作数组costs的索引，以比较不同配方的成本。相对于现有版本，这种代码的效率较高，但理解起来更难！如果你感兴趣，可从wickedlysmart.com下载这些代码。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权



JavaScript填字游戏答案

请完成这个填字游戏，让数组深深地刻在你的脑海中。



5 理解对象

对象镇之旅

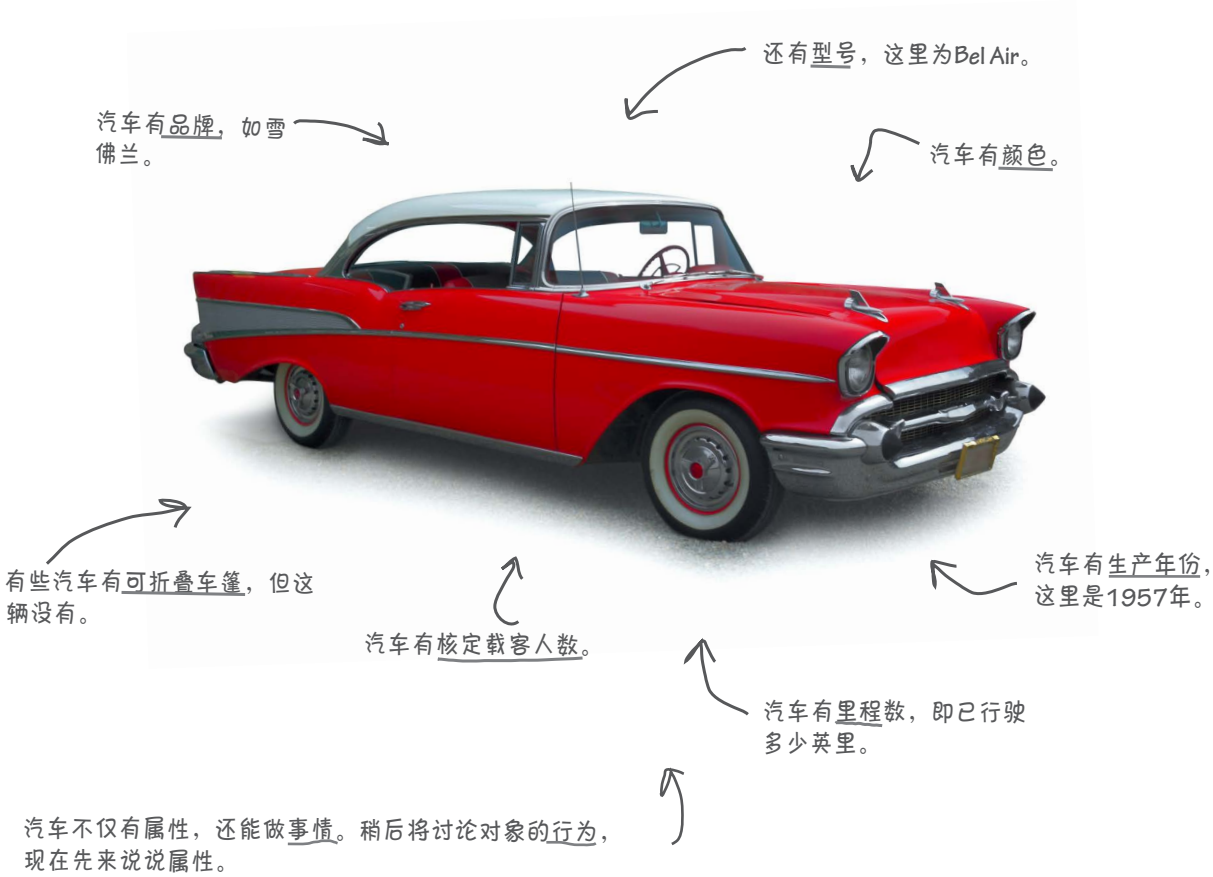


到目前为止，你在代码中使用的都是基本类型和数组。另外，你使用简单语句、条件、for/while循环和函数来编写代码，这种编码方式的**过程化**程度极高，不完全是**面向对象**的。实际上，这根本就不是面向对象的。你确实偶尔在不知不觉间使用了一些对象，但从未编写过自己的对象。现在，该放弃这种枯燥的过程型编程方式，转而创建一些自己的**对象**了。在本章中，你将**了解**为何使用对象可让生活更美好，当然这是从**编程意义**上说的，因为本书只负责提高JavaScript技能，而不能让你变得更时尚。需要指出的是，到了对象镇，你会乐不思蜀，可别忘了给我们寄明信片。

对象

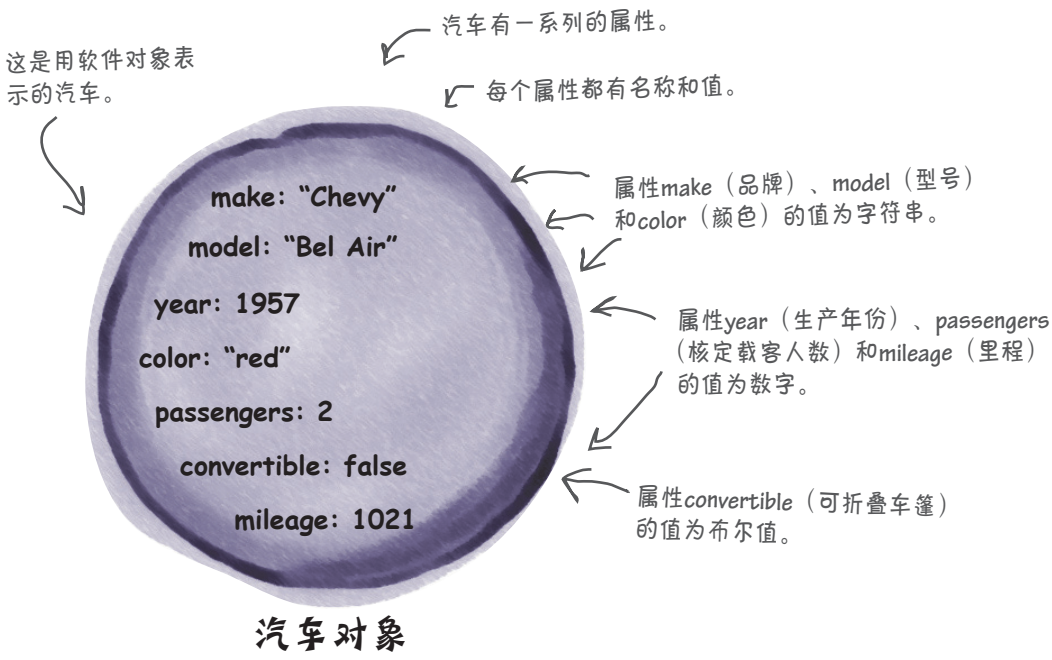
啊，终于到了我们最喜欢的主题！对象（object）将让你的JavaScript编程技能更上一层楼。它们是管理复杂代码、理解浏览器对象模型（这将在下一章介绍）和组织数据的关键，还是很多JavaScript库的基本组织方式（这将在本书后面更详细地介绍）。然而，对象是个不好理解的主题。我们马上就来介绍，不久后你就将使用它们。

不可思议的是，JavaScript对象不过是一系列属性（property）而已。就拿汽车来说吧，它有如下属性。



属性

当然，汽车有很多属性，不仅仅是前面说的那几个，但就编码而言，我们只在软件中记录这几个属性。下面来看看这些属性的JavaScript数据类型。



汽车对象

考考你的脑力

你还希望汽车对象包含其他哪些属性呢？你还能想到汽车的哪些属性呢？将它们都写在下面吧。但别忘了，在软件中只有部分实际属性是有意义的。

那些汽车挂件也许很漂亮，但在对象中记录它们真的有意义吗？

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

磨笔上阵

我们要制作一个表格，在其中列出汽车属性的名称和值。你能帮助我们完成这项任务吗？请将你的答案与本章末尾的答案进行对比，再接着往下阅读。

在这里列出属性的名称。

在这里列出属性的值。

```
{
  make        : "Chevy",
  model       :
  year        :
  color       :
  passengers  :
  convertible :
  mileage     :
              :
              :
};
```

将答案写在这里。你可随意拓展这个列表，在其中添加你想到的其他属性。

请注意指定属性和值时使用的语法，后面可能考查你这方面的知识。

考考你的脑力

出租车有哪些属性和值与前述1957年产的雪佛兰相同？这种车可能有哪些不同之处？它们有哪些额外的属性？又有哪些属性是它们没有的？



如何创建对象

好消息是，在前一个“磨笔上阵”练习中，你已经差不多创建了一个对象；唯一欠缺的工作是将其赋给一个变量，以便能够使用它来做事情，如下所示：

添加一个对象变量声明。

接下来，使用左花括号开启对象定义。

然后，在花括号内定义对象的所有属性。

每个属性的定义都包含名称、冒号和值。这里包含值为字符串、数字和布尔值的属性。

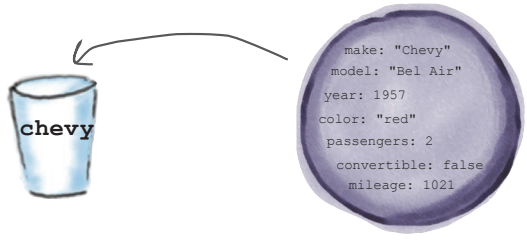
注意到属性之间用逗号分隔。

对象定义以右花括号结尾；与其他变量声明一样，对象变量声明也以分号结束。

```
var chevy = {
    make: "Chevy",
    model: "Bel Air",
    year: 1957,
    color: "red",
    passengers: 2,
    convertible: false,
    mileage: 1021
};
```

这样做的结果是什么呢？当然是一个全新的对象。对象将所有的名称和值（即属性）组合在一起。

至此，你创建了一个包含一系列属性的对象，并将其赋给了一个变量，以便使用它来访问和修改这个对象的属性。



我们根据对象的文本描述创建了一个货真价实的JavaScript对象。

你可以传递这个对象、获取其中的值、修改其中的值、添加属性或删除属性。这此将在稍后介绍，现在再来创建一些对象。

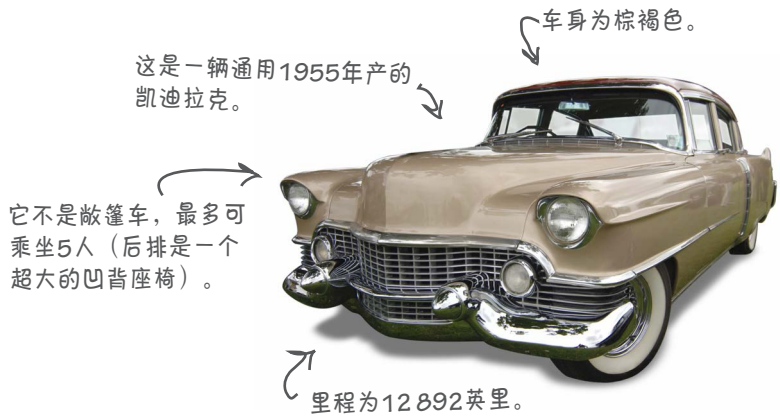


并非只能使用一个对象。稍后你将看到，对象的真正威力在于，你可以创建大量的对象，并编写代码来操作提供给它的对象。来练练手，从头开始创建另一个汽车对象，并将其代码写在下面。

```
var cadi = {

};
```

在这里定义对象的属性。



图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权



超速罚单

签发单位：Web镇警察局

№ 10

说明

别担心，这次算你走运，我们不给你开罚单，只要求你温习下述有关对象创建方面的“交通规则”。

务必将对象定义放在花括号内：

```
var cat = {
    name: "fluffy"
};
```

用冒号分隔属性名和属性值：

```
var planet = {
    diameter: 49528
};
```

属性名可以是任何字符串，但通常遵循变量命名规则：

```
var widget = {
    cost$: 3.14,
    "on sale": true
};
```

请注意，将包含空格的字符串用作属性名时，必须用引号将其括起。

在同一个对象中，不能包含两个同名的属性：

```
var forecast = {
    highTemp: 82,
    highTemp: 56
};
```

不对！这行不通。

用逗号分隔属性名和属性值对：

```
var gadget = {
    name: "anvil",
    isHeavy: true
};
```

在最后一个属性值后面，不添加逗号：

```
var superhero = {
    name: "Batman",
    alias: "Caped Crusader"
};
```

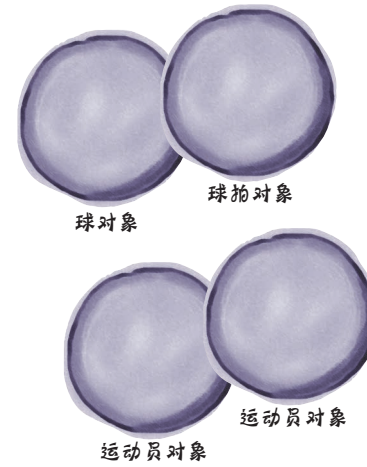
不能在这里添加逗号！

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

何为面向对象呢

到目前为止，我们解决问题时，都使用一系列的变量声明、条件、for/while语句和函数调用。这是**过程型**思维：先这样做，再那样做，等等。在**面向对象**编程中，我们从对象的角度考虑问题，而对象有状态（例如，汽车可能有油位）和行为（例如，汽车可以启动、行驶、停放和停止）。

这是什么意思呢？面向对象编程让你能够解放思想，从更高的层次考虑问题。来看看手工烤面包和使用烤箱烤面包的差别吧。手工烤面包时，需要制作加热线圈，将线圈连接到电源并通电，手持面包放在离线圈很近的地方烤，然后耐心等待，等烤熟后再将线圈断电；使用烤箱时，只需将面包放入烤箱再按下按钮即可。第一种方式是过程型的，而第二种方式是面向对象的：你有一个烤箱对象，让你能够轻松地放入并烤好面包。



你喜欢面向对象的哪些方面？

“它让我能够以更自然的方式进行设计，给事物提供了演进空间。”

——Joy，27岁，软件架构师

“不会打乱已测试过的代码，只需添加新功能即可。”

——Brad，32岁，程序员

“将数据和操作数据的方法都组合到了对象中，我很喜欢。”

——Josh，22岁，产品总监

“在其他应用程序中重用代码。编写新对象时，我能让它足够灵活，从而在新应用程序中使用它。”

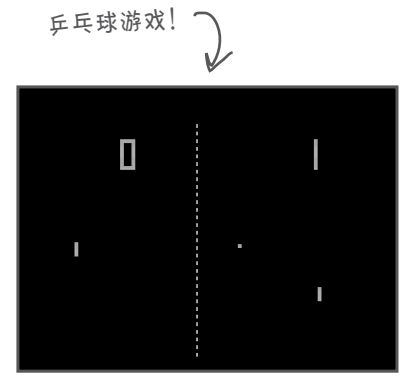
——Chris，39岁，项目经理

“真不敢相信Chris刚才说的，他都5年没编写过一行代码了。”

——Dary，44岁，Chris手下的程序员

考考你的脑力

假设你要编写一个经典的乒乓球街机电子游戏，你将创建哪些对象？你希望这些对象有哪些状态和行为？



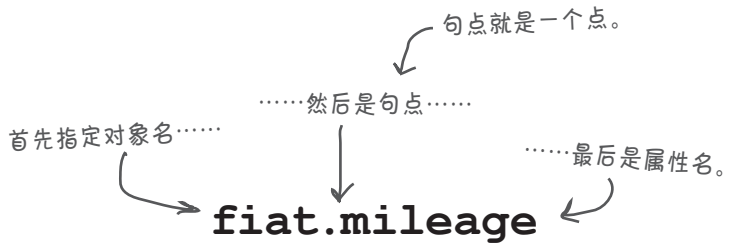



```
var fiat = {
    make: "Fiat",
    model: "500",
    year: 1957,
    color: "Medium Blue",
    passengers: 2,
    convertible: false,
    mileage: 88000
};
```

属性的工作原理

将属性都封装到对象中后，接下来该怎么做呢？你能查看这些属性的值、修改属性的值、添加属性、删除属性，还可使用它们来进行计算。下面来尝试做一些这样的事情——当然是使用JavaScript。

如何访问属性。要访问对象的属性，可依次指定对象名、句点和属性名。这通常被称为句点表示法，看起来类似于下面这样：



可在任何表达式中使用属性，如下所示：

```
var miles = fiat.mileage;
if (miles < 2000) {
    buyIt();
}
```

首先是存储对象的变量，然后是句点和属性名。

句点表示法

- 句点表示法（.）让你能够访问对象的属性。
- 例如，`fiat.color`表示对象`fiat`的一个属性，这个属性的名称为`color`，值为`"Medium Blue"`。

如何修改属性。随时都可修改属性的值，只需将一个新值赋给它即可。例如，假设要将前述菲亚特的里程设置为10 000英里，可以这样做：

```
fiat.mileage = 10000;
```

只需指定要修改的属性，再将一个新值赋给它。请注意，在有些地方，这样做是犯法的。

如何添加新属性。可随时扩展对象，给它们添加新属性。为此，只需指定新属性并给它赋值。例如，假设要给前述菲亚特添加一个布尔属性，用于指出是否该洗车了，可以像下面这样做：

```
fiat.needsWashing = true;
```

如果指定的属性在对象中不存在，将添加该属性；否则将修改该属性的值。



这个新属性被添加到对象中。

如何将属性用于计算。将属性用于计算很简单，只需像使用其他变量（或值）一样使用它们。下面是几个示例：

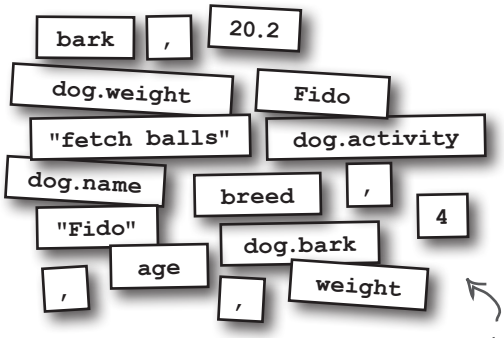
可像使用变量一样使用对象的属性，不同点是需要使用句点表示法来访问对象的属性。

```
if (fiat.year < 1965) {
    classic = true;
}
for (var i = 0; i < fiat.passengers; i++) {
    addPersonToCar();
}
```



对象冰箱贴

下面的代码被胡乱地贴在冰箱上，请利用你的对象创建和句点表示法技能，将它们放在正确的位置。请注意，其中有些冰箱贴可能是多余的！



使用这些冰箱贴来补全下面的代码。



小狗对象

```
var dog = {
    name: __________
    __________: 20.2
    age: _________
    _________: "mixed",
    activity: _____________
};
var bark;
if (_____________ > 20) {
   bark = "WOOF WOOF";
} else {
   bark = "woof woof";
}
var speak = ___________ + " says " + _________ + " when he wants to " + _____________;
console.log(speak);
```



Fido渴望你将其属性归位。



可以，你可随时增删属性。你知道，要给对象添加属性，只需指定一个新属性并给它赋值，如下所示：

```
fido.dogYears = 35;
```

从此之后，fido就有了新属性dogYears。非常简单。

要删除属性，可使用一个特殊的关键字delete。可以像下面这样使用关键字delete：

```
delete fido.dogYears;
```

删除属性时，不仅删除了属性的值，还删除了属性本身。因此，删除属性dogYears后，如果你试图使用fido.dogYears，结果将为undefined。

若成功删除了属性，delete表达式将返回true。仅当属性无法删除时，delete才返回false；这种情况是可能出现的，例如对象属于浏览器而受到保护。即便你要删除的属性在对象中不存在，delete也将返回true。

世上没有愚蠢的问题

问：一个对象最多可包含多少个属性？

答：想要多少就可以有多少。对象可以没有任何属性，也可以有数百个属性，完全由你决定。

问：如何创建没有属性的对象？

答：像创建其他对象一样，只是不指定任何属性，就像下面这样：

```
var lookMaNoProps = { };
```

问：我知道，我刚问了如何创建没有属性的对象，可为何要这样做呢？

答：根据代码的逻辑，你可能想先创建一个空对象，再动态地添加属性。随着你不断使用对象，你将明白为何要采用这种创建对象的方式。

```
var lookMaNoProps = { };
lookMaNoProps.age = 10;
if (lookMaNoProps.age > 5) {
  lookMaNoProps.school = "Elementary";
}
```

问：与使用一系列变量相比，使用对象有何优势？毕竟对于前述菲亚特对象的每个属性，都可将其作为独立的变量，不是吗？

答：对象封装了数据的复杂性，让你能够专注于代码的高层次设计，而不是细枝末节。假设你要编写一个设计10辆车的交通模拟器，你肯定希望专注于汽车、路灯和马路，而不是数百个变量。对象还让你的工作更轻松，因为它们封装（隐藏）对象状态和行为的复杂性，让你无需操心这些东西。这些都是如何实现的呢？等你有了更多对象方面的经验后，就会明白得多。

问：试图给对象添加既有的属性时，结果将如何？

答：试图给对象添加既有属性（如`needsWashing`）时，将修改该属性的值。因此，如果你编写了代码：

```
fiat.needsWashing = true;
```

而`fiat`包含值为`false`的属性`needsWashing`，这将把这个属性的值改为`true`。

问：试图访问不存在的属性时，结果将如何？例如，如果我编写了代码`if (fiat.make) { ... }`，而`fiat`没有属性`make`，结果将如何呢？

答：如果`fiat`没有属性`make`，表达式`fiat.make`的结果将为`undefined`。

问：如果在最后一个属性值后面加上了逗号，结果将如何？

答：在大多数浏览器中，这不会导致错误。然而，在一些浏览器的旧版本中，这会导致JavaScript停止执行。因此，要确保代码能够在尽可能多的浏览器中运行，千万不要添加多余的逗号。

问：可以使用`console.log`来将对象显示到控制台吗？

答：可以，为此只需编写类似于下面的代码：

```
console.log(fiat);
```

这样，当你加载网页时，如果控制台打开了，你将在控制台中看到有关指定对象的信息。

JavaScript控制台

```
> console.log(fiat)
Object {make: "Fiat", model: "500", year: 1957,
color: "Medium Blue", passengers: 2…}
>
```

变量是如何存储对象的
爱寻根究底的人想知道……



幕后花絮

你知道，变量像是容器，用于存储值。数字、字符串和布尔值都很小，对象怎么样呢？变量可存储任何规模的对象，不管它包含多少属性吗？

- 变量并不实际存储对象。
- 变量存储指向对象的引用。
- 引用就像指针，是对象的存储地址。
- 换句话说，变量并不存储对象本身，而是存储类似于指针的东西。在JavaScript中，我们并不知道引用变量存储的到底是什么，但我们知道，不管它存储的是什么，它肯定指向相应的对象。
- 当我们使用句点表示法时，JavaScript解释器将负责根据引用获取对象并访问其属性。

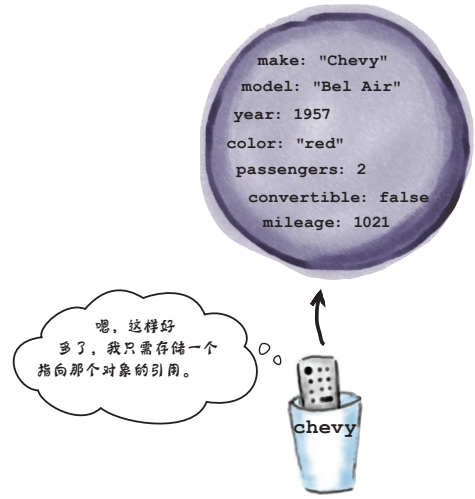


因此，虽然我们通常认为对象是被塞到变量中的，但这是不可能的，就像根本不存在可伸缩的庞大杯子，能够装下任何东西一样。相反，对象变量存储的是指向对象的引用。

你也可以这样想：基本类型变量表示的是实际**值**，而对象变量表示一种**获取对象的途径**。在实际工作中，你只需将对象视为像狗和汽车一样的东西（而不是引用），但知道变量存储的是指向对象的**引用**将在本书后面派上用场（再过几页你就会看到）。

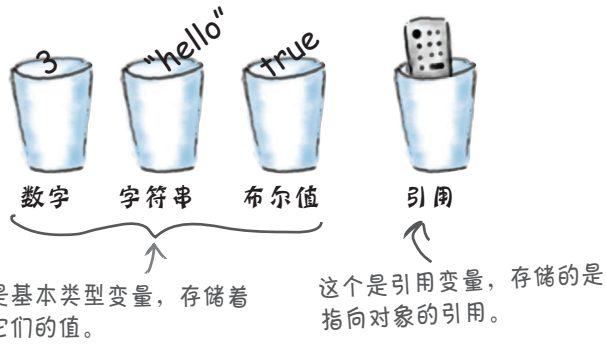
另外，当你将句点表示法（.）用于引用变量时，相当于这样说：请使用句点**前**的引用来获取相应的对象，再访问句点**后**指定的属性。（请多念几遍，将其完全消化。）例如，下述代码的含义是，访问变量car引用的对象的属性color。

```
car.color;
```



比较基本类型和对象

可将对象引用视为另一个变量值，这意味着引用就像基本类型一样，也是可以放入杯子的。基本类型变量存储的是实际**值**，如5、26.7、"hi"或false；而引用变量存储的是**引用**：一种获取特定对象的途径。





幕后花絮

我们不知道（也不关心）JavaScript解释器如何表示对象引用。

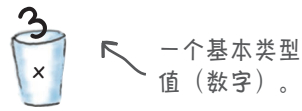
我们只知道可以使用句点表示法来访问对象的属性。

初始化基本类型变量

声明并初始化基本类型变量时，你赋给它一个值；这个值放在变量这个杯子中，如下所示：

```
var x = 3;
```

这个变量存储着数字3。



初始化对象（引用）变量

声明并初始化对象变量时，你使用对象表示法创建一个对象，但这个对象在变量这个杯子中放不下，因此实际放到杯子里的是指向这个对象的**引用**。

```
var myCar = {...};
```

变量中存储的是指向汽车对象的引用。

汽车对象本身并没有存储到变量中。



图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

使用对象



假设你要物色一辆好车，供你在Web镇使用。你的要求是什么呢？下面的要求怎样？

- ❑ 1960年或更早生产。
- ❑ 里程不超过10 000英里。

你还想利用新学到的编码技能来简化挑选工作。因此你想编写一个函数，帮你对车辆进行预检，即如果车辆合乎要求，这个函数就返回`true`；如果车辆不值得你浪费时间，这个函数就返回`false`。

具体地说，你要编写一个**函数**，它**将一个汽车对象作为参数**，对其进行检查，并返回一个布尔值。这个函数可用于对任何汽车对象进行预检。

下面来尝试编写这个函数：

这是所需的函数。

你将给它传递一个汽车对象。

```
function prequal(car) {
    if (car.mileage > 10000) {
        return false;
    } else if (car.year > 1960) {
        return false;
    }
    return true;
}
```

将句点表示法用于形参car，以访问属性mileage和year。

检查每个属性的值，看它们是否符合要求。

只要有一项要求没有满足，就返回false；否则返回true，表示车辆通过了预检。

下面来尝试使用这个函数。为此，先得有一个汽车对象，下面这个怎样？

```
var taxi = {
    make: "Webville Motors",
    model: "Taxi",
    year: 1955,
    color: "yellow",
    passengers: 4,
    convertible: false,
    mileage: 281341
};
```

你觉得怎样？该考虑这辆黄色出租车吗？请说明理由。



进行预检



关于对象说得够多了，下面来创建一个，并使用函数prequal对其进行预检。在一个简单的HTML页面（prequal.html）中，嵌入下面的代码，然后加载这个网页，看看这辆出租车是否合乎要求：

```
var taxi = {
    make: "Webville Motors",
    model: "Taxi",
    year: 1955,
    color: "yellow",
    passengers: 4,
    convertible: false,
    mileage: 281341
};

function prequal(car) {
    if (car.mileage > 10000) {
        return false;
    } else if (car.year > 1960) {
        return false;
    }
    return true;
}

var worthALook = prequal(taxi);

if (worthALook) {
    console.log("You gotta check out this " + taxi.make + " " + taxi.model);
} else {
    console.log("You should really pass on the " + taxi.make + " " + taxi.model);
}
```

这辆出租车合乎要求吗

这是我们得到的输出。下面来分步查看这些代码，看看这辆出租车没有通过预检的原因。

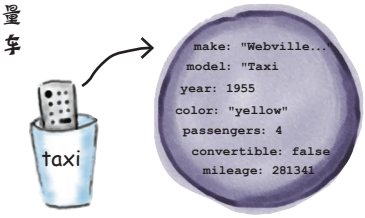
JavaScript控制台

```
You should really pass on the Webville Motors Taxi
```

分步进行预检

① 首先，我们创建了出租车对象，并将其赋给变量taxi。当然，变量taxi存储的是指向这个出租车对象的引用，而不是对象本身。

```
var taxi = { ... };
```



② 接下来，我们调用函数prequal，并传入参实taxi，该实参将被赋给函数形参car。

```
function prequal(car) {
    ...
}
```

car和taxi指向同一个对象。



③ 接下来，我在函数体中对形参car指向的出租车对象进行预检。

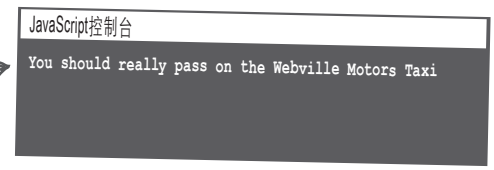
```
if (car.mileage > 10000) {
    return false;
} else if (car.year > 1960) {
    return false;
}
```



这辆出租车的里程远高于10000英里，因此函数prequal返回false。真遗憾，坐起来非常惬意。

④ 真遗憾，这辆出租车的里程太长，因此第一个测试（car.mileage > 10000）为true。函数返回false，因此worthALook被设置为false。接下来，控制台中显示了You should really pass on the Webville Motors Taxi。

函数prequal返回false，因此控制台输出为……




```
var worthALook = prequal(taxi);

if (worthALook) {
    console.log("You gotta check out this " + taxi.make + " " + taxi.model);
} else {
    console.log("You should really pass on the " + taxi.make + " " + taxi.model);
}
```



磨笔上阵

轮到你了。这里还有三个汽车对象；将它们分别传递给函数prequal时，结果如何呢？请先通过心算来回答，再编写代码来检查你的答案是否正确。



```
var cadi = {
    make: "GM",
    model: "Cadillac",
    year: 1955,
    color: "tan",
    passengers: 5,
    convertible: false,
    mileage: 12892
};

prequal(cadi);
```

_____

在这里填写函数prequal返回的值。



```
var fiat = {
    make: "Fiat",
    model: "500",
    year: 1957,
    color: "Medium Blue",
    passengers: 2,
    convertible: false,
    mileage: 88000
};

prequal(fiat);
```

_____



```
var chevy = {
    make: "Chevy",
    model: "Bel Air",
    year: 1957,
    color: "red",
    passengers: 2,
    convertible: false,
    mileage: 1021
};

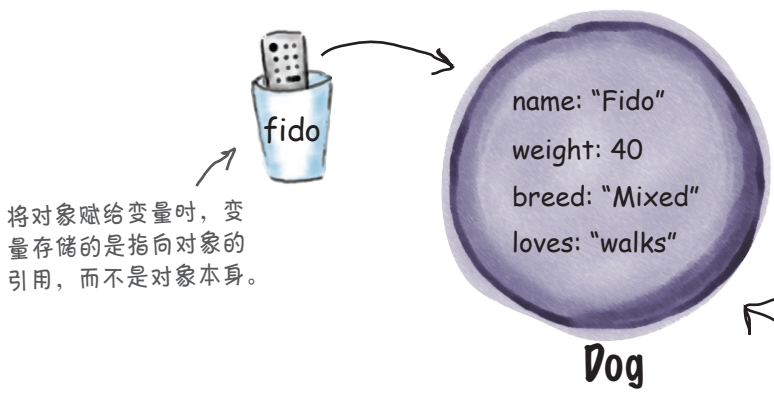
prequal(chevy);
```

_____

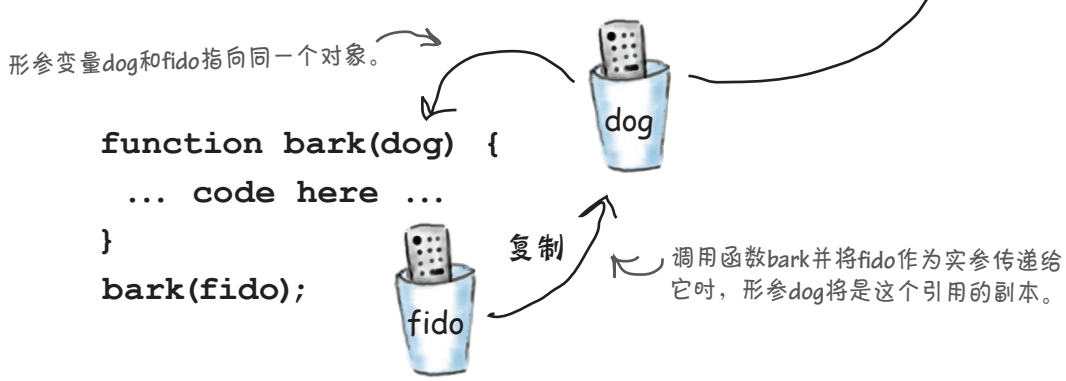
更深入地讨论向函数传递对象的情况

前面简要地讨论了实参是如何传递给函数的：实参是**按值传递**的，这意味着传递的是实参的副本。因此，如果我们传递一个整数，相应的函数形参将获得该整数值的副本，以便在函数中使用它。传递对象时情况亦如此，但我们必须更深入地研究按值传递对对象来说意味着什么，这样才能明白向函数传递对象时发生的情况。

你知道，将对象赋给变量时，变量存储的是指向对象的**引用**，而不是对象本身。再次重申，可将引用视为指向对象的指针。



因此，调用函数并向它传递一个对象时，传递的是**对象引用**，而不是对象本身。根据按值传递的语义，传递给形参的是该引用的副本，而原来的引用依然是指向原始对象的指针。

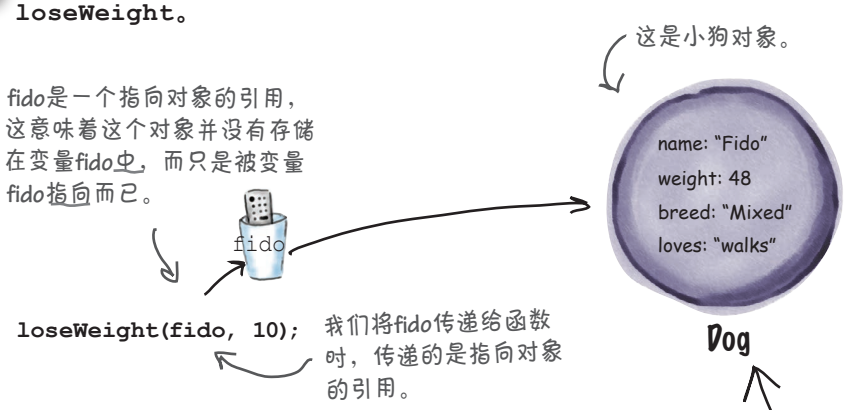


这意味着什么呢？一个最大的不同是，如果你在函数中修改对象的属性，修改的将是原始对象的属性。因此，函数结束时，在函数中对对象所做的修改都依然有效。下面通过一个示例来详细说明这一点。

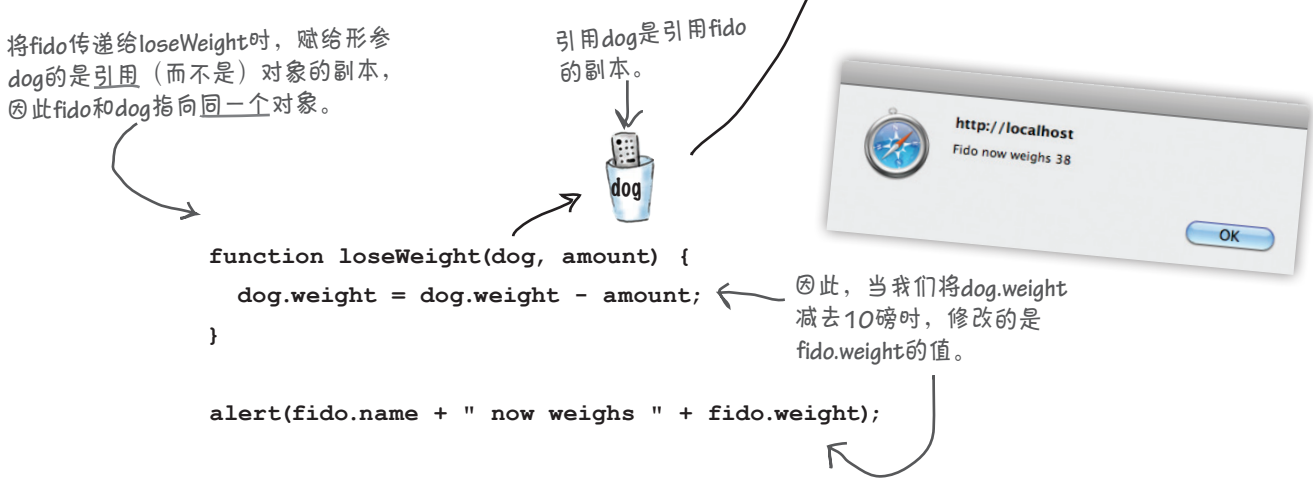
让fido减减肥

假设我们要测试一种给小狗瘦身的新方法，并要在一个名为loseWeight的函数中实现这种减肥方法。你只要将小狗对象传递给函数loseWeight，并指定要减去多少，小狗的体重就会减下来，就像变魔术一样。下面是其中的工作原理。

① 首先来看看小狗对象fido，我们将把它传递给函数loseWeight。



② 函数loseWeight的形参dog是fido存储的引用的副本，因此修改这个形参变量的属性时，将影响传入的引用指向的对象。



图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

磨笔上阵



有人给你提供了一个绝密文件，还有两个让你能够获取和设置该文件内容的函数，但要使用这两个函数，你必须知道正确的密码。第一个函数是getSecret，它在密码正确时返回文件的内容，并将每次访问尝试都记录在案。第二个函数是setSecret，它修改文件的内容，并将访问次数重置为零。你的任务是补全下面的JavaScript代码，并对完成后的函数进行测试。

```
function getSecret(file, secretPassword) {
    ________.opened = ________.opened + 1;
    if (secretPassword == ________.password) {
        return _______.contents;
    }
    else {
        return "Invalid password! No secret for you.";
    }
}
function setSecret(file, secretPassword, secret) {
    if (secretPassword == ________.password) {
        _______.opened = 0;
        _______.contents = secret;
    }
}

var superSecretFile = {
    level: "classified",
    opened: 0,
    password: 2,
    contents: "Dr. Evel's next meeting is in Detroit."
};
var secret = getSecret(__________________, ______);
console.log(secret);

setSecret(__________________, ______, "Dr. Evel's next meeting is in Philadelphia.");
secret = getSecret(__________________, ______);
console.log(secret);
```


图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权




```
<!doctype html>
<html lang="en">
<head>
  <title>Object-o-matic</title>
  <meta charset="utf-8">
  <script>
    function makeCar() {
        var makes = ["Chevy", "GM", "Fiat", "Webville Motors", "Tucker"];
        var models = ["Cadillac", "500", "Bel-Air", "Taxi", "Torpedo"];
        var years = [1955, 1957, 1948, 1954, 1961];
        var colors = ["red", "blue", "tan", "yellow", "white"];
        var convertible = [true, false];

        var rand1 = Math.floor(Math.random() * makes.length);
        var rand2 = Math.floor(Math.random() * models.length);
        var rand3 = Math.floor(Math.random() * years.length);
        var rand4 = Math.floor(Math.random() * colors.length);
        var rand5 = Math.floor(Math.random() * 5) + 1;
        var rand6 = Math.floor(Math.random() * 2);

        var car = {
            make: makes[rand1],
            model: models[rand2],
            year: years[rand3],
            color: colors[rand4],
            passengers: rand5,
            convertible: convertible[rand6],
            mileage: 0
        };
        return car;
    }

    function displayCar(car) {
        console.log("Your new car is a " + car.year + " " + car.make + " " + car.model);
    }

    var carToSell = makeCar();
    displayCar(carToSell);

  </script>
</head>
<body></body>
</html>
```

汽车自动制造器与第4章的自动造句应用程序类似，但其中的单词表示的是汽车属性，并且生成的是汽车对象，而不是广告语！

看看它的功能和工作原理吧。

汽车自动制造机

推销自动造句应用程序的那个家伙又来推销自动汽车制造机（Auto-O-Matic）了，这种机器可整天不间断地生产已停产的汽车。换句话说，它不生成广告语，而是生成品牌、型号、年份等汽车对象的属性，犹如汽车工厂。下面来详细看看其工作原理。

① 首先是一个makeCar函数，我们可随时调用它来生产新汽车。在这个函数中，有4个表示汽车品牌、型号、生产年份和颜色的数组，还有一个表示汽车是否有可折叠车篷的数组。我们生成5个随机数，以便从这5个数组中随机选择品牌、型号、生产年份、颜色以及是否有可折叠车篷。我们还生成了另一个随机数，用于表示核定载客人数。

我们可从这4个数组中选择多种品牌、型号、生产年份和颜色。

```
var makes = ["Chevy", "GM", "Fiat", "Webville Motors", "Tucker"];
var models = ["Cadillac", "500", "Bel-Air", "Taxi", "Torpedo"];
var years = [1955, 1957, 1948, 1954, 1961];
var colors = ["red", "blue", "tan", "yellow", "white"];
var convertible = [true, false];
```

我们将从这个数组中选择属性convertible的值（true或false）。

我们使用这4个随机数从数组中随机地选择值。

```
var rand1 = Math.floor(Math.random() * makes.length);
var rand2 = Math.floor(Math.random() * models.length);
var rand3 = Math.floor(Math.random() * years.length);
var rand4 = Math.floor(Math.random() * colors.length);
var rand5 = Math.floor(Math.random() * 5) + 1;
var rand6 = Math.floor(Math.random() * 2);
```

我们将这个随机数作为核定载客人数。将随机数加上1，确保汽车的核定载客人数不少于1。

我们根据这个随机数确定汽车是否有可折叠车篷。

② 这里不像自动造句应用程序那样将各种汽车属性拼接成一个字符串，而是使用它们来创建新对象car，它有你期望的所有属性。为了从数组中选择属性make、model、year和color的值，我们使用了第1步创建的随机数，我们还添加了passengers、convertible和mileage属性。

```
var car = {
    make: makes[rand1],
    model: models[rand2],
    year: years[rand3],
    color: colors[rand4],
    passengers: rand5,
    convertible: convertible[rand6],
    mileage: 0
};
```

我们创建一个新的汽车对象，其属性的值是从数组中选择出来的。

我们还将核定载客人数设置为前面创建的一个随机数，并使用数组convertible将属性convertible设置为true或false。

最后，将属性mileage设置为0（因为这是一辆新车）。

③ 函数makeCar的最后一条语句返回新对象car。

```
return car;
```

从函数返回对象与返回其他值没什么两样。下面来看看调用函数makeCar的代码：

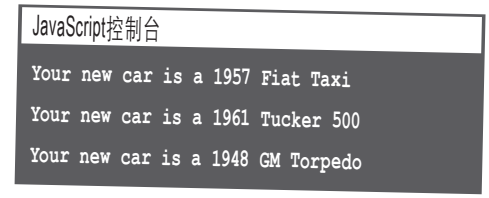
```
function displayCar(car) {
  console.log("Your new car is a " + car.year + " " +
              car.make + " " + car.model);
}
var carToSell = makeCar();
displayCar(carToSell);
```

别忘了，你返回了一个指向汽车对象的引用，并将其赋给了变量carToSell。



首先调用函数makeCar，并将它返回的值赋给变量carToSell。接下来，我们将makeCar返回的汽车对象传递给函数displayCar，它所做的只是在控制台中显示这个对象的一些属性。

④ 请在浏览器中加载这个自动汽车制造机（autoomatic.html），让它运转起来。你将发现，你想生成多少新车就能生产多少，但别忘了，“每分钟都有笨蛋出生”。



这是你的新车。在我们看来，1957年产的菲亚特出租车很不错。

像我们这样多次重新加载这个网页！

如何给对象添加行为

你不会认为对象只能用于存储数字和字符串吧？对象是**活动的**，能够**做事情**。小狗不会一直静静地坐着，它们还会叫、跑、玩接球游戏。小狗对象也如此！同理，我们可以开车、停车、倒车和刹车。利用本章介绍过的知识，你完全能够给对象添加行为，如下所示：

```
var fiat = {
    make: "Fiat",
    model: "500",
    year: 1957,
    color: "Medium Blue",
    passengers: 2,
    convertible: false,
    mileage: 88000,
    drive: function() {
       alert.log("Zoom zoom!");
    }
};
```

可以像这样直接给对象添加函数。

你需要做的只是将函数定义赋给属性。是的，属性也可以是函数！

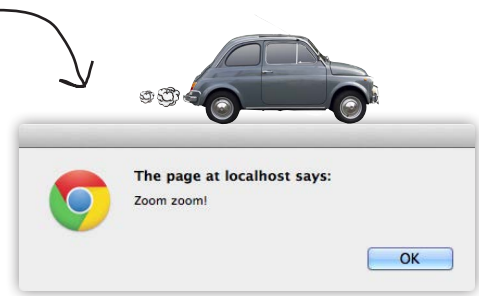
注意到在函数定义中没有指定函数名，而只是在关键字function后面提供了函数体。属性名就是函数名。

稍微说说术语的问题：对于对象内的函数，我们通常称之为方法。在面向对象领域，将对象内的函数统称为方法。

要调用函数drive，准确地说是方法drive，也使用句点表示法：用句点将对象名（fiat）和属性名（drive）连接起来，并在属性名后面加上圆括号，就像调用其他函数一样。

```
fiat.drive();
```

我们使用句点表示法来访问对象fiat内的函数，就像访问其他属性一样。这条语句的含义是调用对象fiat的方法drive。



调用对象fiat的方法drive的结果。

改进方法drive

下面来让fiat对象的行为更像汽车。对大多数汽车来说，只有启动发动机后才能开动。如何模拟这种行为呢？为此，需要：

- 一个布尔属性，用于存储汽车的状态（发动机是否开启）；
- 两个方法，分别用于启动和熄灭发动机；
- 在方法drive中检查条件，确保仅当发动机启动后才能开动汽车。

下面首先添加了布尔属性started以及启动和熄灭发动机的方法start和stop，再修改方法drive，使其根据属性started决定是否开动汽车。

```
var fiat = {
    make: "Fiat",
    model: "500",
    year: 1957,
    color: "Medium Blue",
    passengers: 2,
    convertible: false,
    mileage: 88000,
    started: false,

    start: function() {
       started = true;
    },

    stop: function() {
        started = false;
    },

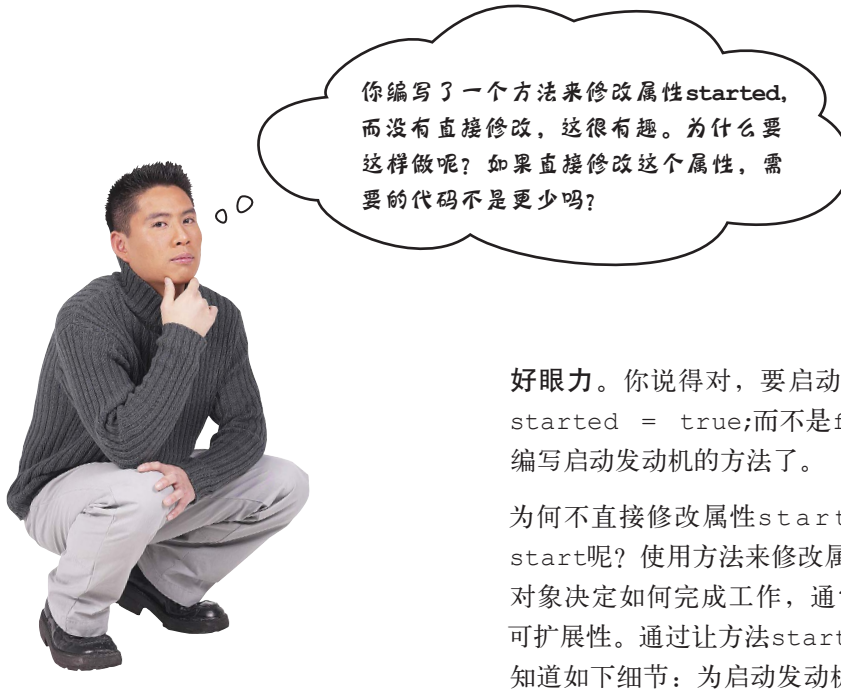
    drive: function() {
        if (started) {
            alert("Zoom zoom!");
        } else {
            alert("You need to start the engine first.");
        }
    }
};
```

存储发动机当前状态的属性（true表示发动机已启动，false表示发动机已熄灭）。

启动发动机的方法，当前它只是将属性started设置为true。

熄灭发动机的方法，它只是将属性started设置为false。

在这里实现了有趣的行为：当你试图开动汽车时，如果发动机已启动，将显示Zoom zoom!，否则你将被告知要先启动发动机。



好眼力。你说得对，要启动发动机，可使用代码`fiat.started = true;`而不是`fiat.start();`。这样就无需编写启动发动机的方法了。

为何不直接修改属性`started`，而要创建并调用方法`start`呢？使用方法来修改属性是另一种封装方式，它让对象决定如何完成工作，通常可改善代码的可维护性和可扩展性。通过让方法`start`负责启动发动机，你就无需知道如下细节：为启动发动机，需要获取属性`started`，并将其设置为`true`。

你可能还是会说，这有什么大不了的，为何不直接将这个属性设置为`true`来启动发动机呢？假设有一个非常复杂的`start`方法，需要确定安全带已系好、确保油料充足、检查电瓶、检查发动机温度等，然后再将`started`设置为`true`。你肯定不想在开车前考虑所有这些细节，而希望能够调用一个方法来完成这些工作。通过将这些细节都放在一个方法中，让你只需向对象发出指令，由对象去操心如何执行这些指令即可。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

测试对象fiat



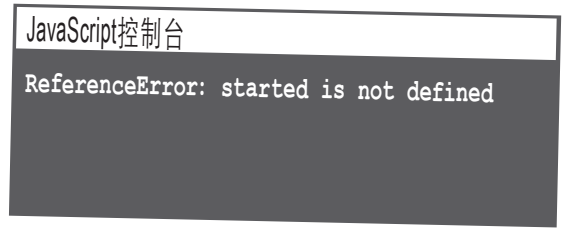
来测试一下改进后的对象fiat。我们将进行全面测试：在没有启动发动机的情况下尝试开动，再启动发动机、开车并熄火。为此，在一个简单的HTML页面（carWithDrive.html）中嵌入对象fiat的代码，再在这些代码后面输入如下代码：

```
fiat.drive();   ← 首先，我们尝试开动汽车，这将显示一
fiat.start();   ← 条消息，让我们先启动发动机。然后，
fiat.drive();   ← 我们启动发动机，并开动汽车。最后，
fiat.stop();    ← 我们熄灭发动机。
```

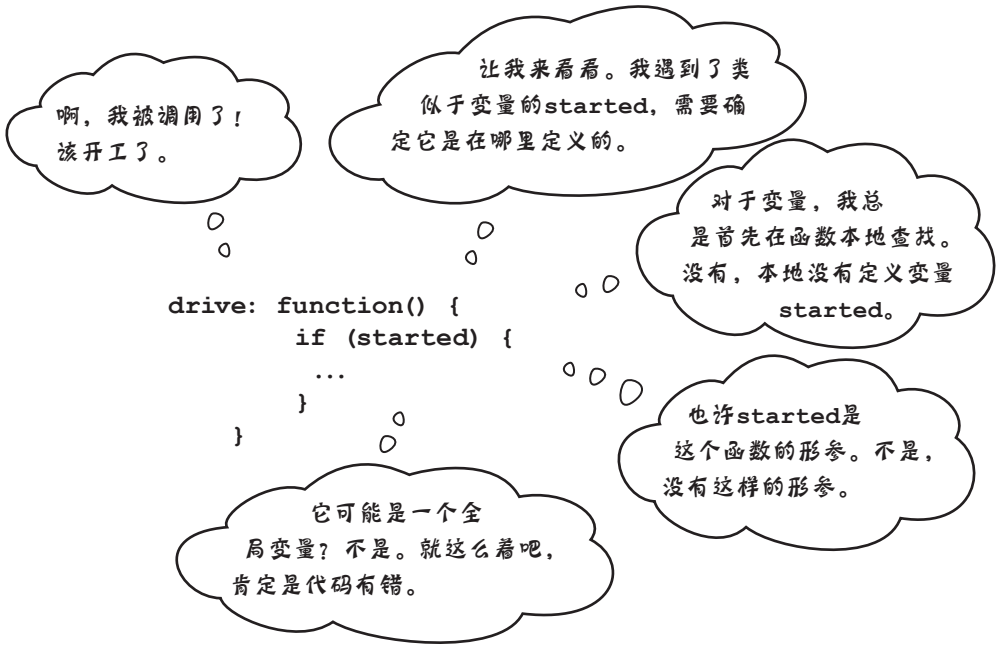
在浏览器中加载这个网页，开始一次自驾游吧！

别着急

如果你开不动汽车对象fiat也很正常。实际上，如果打开JavaScript控制台，你很可能看到一条类似于后面的消息，指出started未定义。



到底是怎么回事呢？咱们来仔细研究方法drive，看看当我们使用fiat.drive()试图开动这辆汽车时发生的情况。



图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

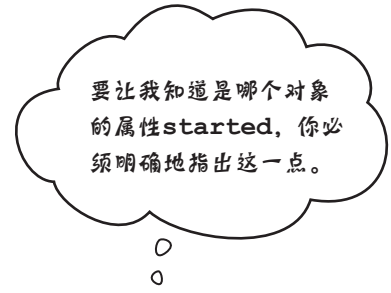
为何方法drive不知道属性started

原因如下：在对象fiat的方法中，我们原本要引用属性started，但函数中引用的变量通常被解析为局部变量、函数形参或全局变量。在方法drive中，started不属于上述任何一种变量，而是对象fiat的一个**属性**。

虽然如此，代码也该能正确地运行吧？换句话说，我们在对象fiat中定义了started，JavaScript应该足够聪明，知道我们指的是属性started吧？

正如你看到的，它不知道。怎么会这样呢？

情况是这样的：在方法drive中，看起来像变量的started实际上是对象的一个属性，但我们没有告诉JavaScript它是哪个对象的属性。你可能会说，我们说的显然是当前对象，这不是明摆着的吗！有什么不清楚的呢？是的，我们的本意是当前对象的属性。事实上，有一个JavaScript关键字this，可使用它来告诉JavaScript，你说的是**当前所处的对象**。




```
drive: function() {
         if (started) {
           ...
         }
       }
```

下面来加上关键字this，让这些代码能够正常运行：

```
var fiat = {
    make: "Fiat",
    // 其他属性。为节省篇幅，这里省略了它们
    started: false,

    start: function() {
        this.started = true;
    },

    stop: function() {
        this.started = false;
    },

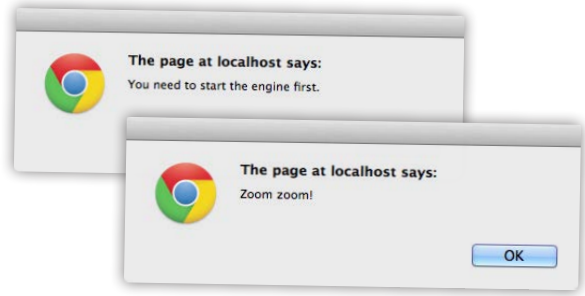
    drive: function() {
        if (this.started) {
            alert("Zoom zoom!");
        } else {
            alert("You need to start the engine first.");
        }
    }
};
```

每次引用属性started时，都在它前面加上this和句点，告诉JavaScript解释器你说的是当前对象的属性started，以免JavaScript以为你引用的是一个变量。

测试关键字this



请修改你的代码，并尝试运行它们！
我们得到的结果如下。



变身浏览器



下面的JavaScript代码存在一些错误，你的任务是变身浏览器，将这些错误找出来。完成这个练习后，请翻到本章末尾，看看你是否找出了所有的错误。

在这里标出代码中的错误。

```
var song = {
    name: "Walk This Way",
    artist: "Run-D.M.C.",
    minutes: 4,
    seconds: 3,
    genre: "80s",
    playing: false,

    play: function() {
        if (!playing) {
            this = true;
            console.log("Playing "
                + name + " by " + artist);
        }
    },

    pause: function() {
        if (playing) {
            this.playing = false;
        }
    }
};

this.play();
this.pause();
```

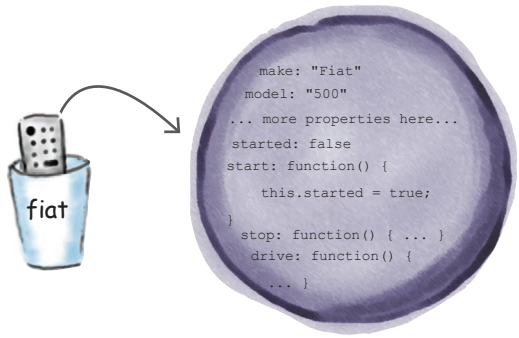

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

关键字this的工作原理



可将this视为一个变量，指向其方法被调用的对象。换句话说，如果你使用fiat.start()调用对象fiat的方法start，并在方法start中使用this，那么this将指向对象fiat。下面来更深入地研究调用对象fiat的方法start时发生的情况。

首先，我们有一个表示菲亚特汽车的对象，并将其赋给了变量fiat。



这是菲亚特汽车对象，其中包含所有的属性名和属性值，还有方法start。

接下来，我们调用方法start，JavaScript负责将对象fiat赋给this。

在这里，this指向对象fiat，因为我们调用的是对象fiat的方法start。

```
fiat.start();
```

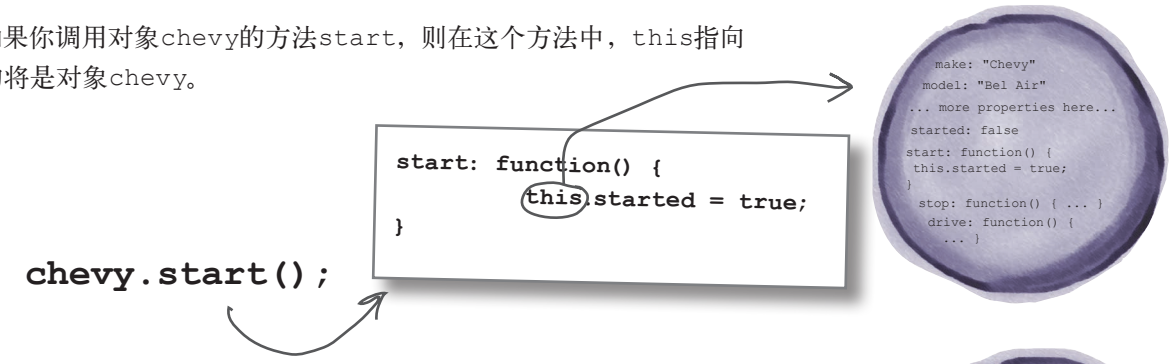
```
start: function() {
         this.started = true;
}
```

每当我们调用对象的方法时，this都将指向这个对象。因此，在这里this指向对象fiat。

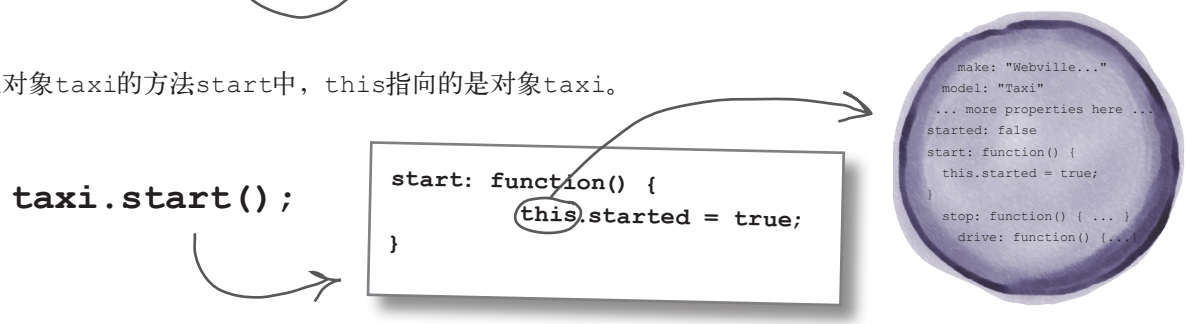
要理解关键字this，关键在于每当方法被调用时，在该方法体内都可使用this来引用**方法被调用的对象**。为了帮助你理解透彻，来尝试调用其他几个对象的方法。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权

如果你调用对象chevy的方法start，则在这个方法中，this指向的将是对象chevy。



在对象taxi的方法start中，this指向的是对象taxi。



磨笔上阵

请利用新学到的有关this的知识补全下面的代码，答案见本章末尾。

```
var eightBall = { index: 0,
                  advice: ["yes", "no", "maybe", "not a chance"],
                  shake: function() {
                             this.index = _________.index + 1;
                             if (__________.index  >= __________.advice.length) {
                                 __________.index = 0;
                             }
                  },
                  look: function() {
                             return __________.advice[__________.index];
                  }
};
eightBall.shake();
console.log(eightBall.look());
```

重复执行这些代码多次，对你补全的代码进行测试。

JavaScript控制台

```
no
maybe
not a chance
```

世上没有愚蠢的问题

问：方法和函数有何不同？

答：方法是在对象中赋给了属性名的函数。你通过函数名来调用函数，而调用方法时，你使用对象句点表示法和属性名。在方法中，你还可使用关键字this来引用方法被调用的对象。

问：我注意到在对象中使用关键字function时，没有显式地给函数指定名称。这种函数的名称是什么呢？

答：是这样的。为调用方法，我们使用对象中相应的属性名，因此无需显式地给这种函数命名，以便用来调用它。就现在而言，将此视为一种约定即可，本书后面将深入探讨匿名函数，即未被显式命名的函数。

问：函数可以有局部变量，方法也可以吗？

答：可以。方法其实就是函数，将其称为方法只是因为它们位于对象中。因此，函数能做的，方法都能做，因为方法就是函数。

问：那么，也可以从方法返回值吗？

答：可以。看看我们在上一个回答中是怎么说的！

问：也可以向方法传递实参吗？

答：可以！难道你没有看前两个问题的答案吗？

问：创建对象后，可以向添加属性一样添加方法吗？

答：可以。可将方法视为被赋给属性的函数，因此可随时添加新方法：

```
// 添加方法engageTurbo
car.engageTurbo =
        function() { ... };
```

问：在添加的方法（如前面的engageTurbo）中，也可以使用关键字this吗？

答：可以。别忘了，调用方法时，将把方法被调用的对象赋给this。

问：this的值在什么时候被设置为相应的对象？是在定义对象时还是调用方法时？

答：在你调用方法时，this的值被设置为相应的对象。因此，当你调用fiat.start()时，this被设置为fiat，而当你调用chevy.start()时，this被设置为chevy。看起来好像this是在你定义对象时被设置的，因为在fiat.start中，this总是被设置为fiat，而在chevy.start中，this总是被设置为chevy，但你将在本书后面看到，有充分的理由表明this是在你调用方法而不是定义对象时设置的。这一点很重要，我们将反复讨论。

考考你的脑力

如果你将方法start、stop和drive复制到前面创建的对象chevy和cadi中，必须如何修改才能让它们正确地工作？

答案：根本不用修改！因为this指的是当前对象，即其方法被调用的对象。



该把整个车队拉出来“遛遛”了。给每个汽车对象都添加方法drive，然后添加对它们进行发动、驾驶和熄火的代码。答案见本章末尾。



```
var cadi = {
    make: "GM",
    model: "Cadillac",
    year: 1955,
    color: "tan",
    passengers: 5,
    convertible: false,
    mileage: 12892
};
```



```
var chevy = {
    make: "Chevy",
    model: "Bel Air",
    year: 1957,
    color: "red",
    passengers: 2,
    convertible: false,
    mileage: 1021
};
```



```
var taxi = {
    make: "Webville Motors",
    model: "Taxi",
    year: 1955,
    color: "yellow",
    passengers: 4,
    convertible: false,
    mileage: 281341
};
```

添加新属性时，别忘了在属性mileage后面加上逗号！

给每个汽车对象都添加属性started和下面的方法，再使用下面的代码来测试它们。

```
started: false,

start: function() {
   this.started = true;
},

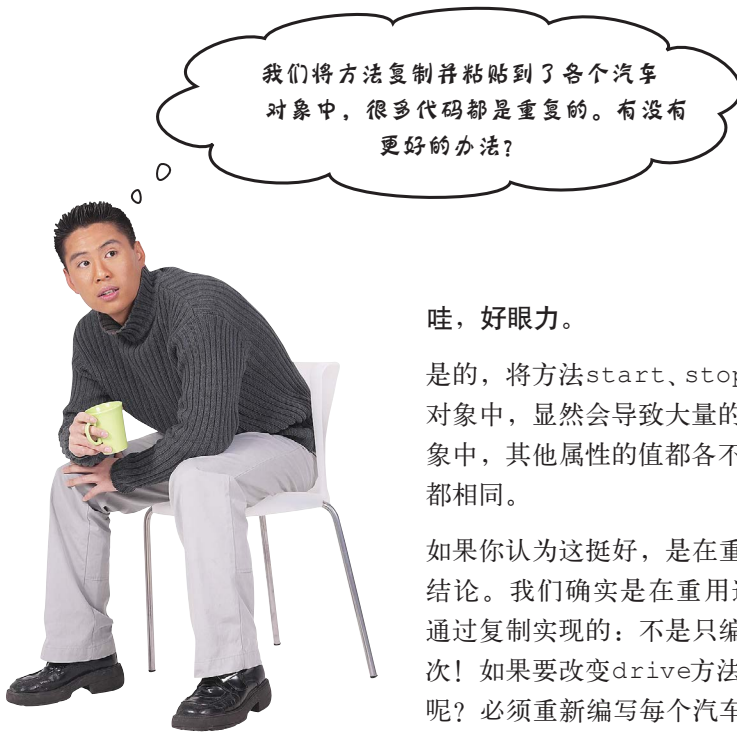
stop: function() {
   this.started = false;
},

drive: function() {
   if (this.started) {
     alert(this.make + " " +
           this.model + " goes zoom zoom!");
   } else {
     alert("You need to start the engine first.");
   }
}
```

我们稍微改进了方法drive，请你务必使用这些新代码。

在汽车对象定义后面添加这些代码，对所有对象都进行测试。

```
cadi.start();
cadi.drive();
cadi.stop();
chevy.start();
chevy.drive();
chevy.stop();
taxi.start();
taxi.drive();
taxi.stop();
```



哇，好眼力。

是的，将方法start、stop和drive复制到每个汽车对象中，显然会导致大量的重复代码。在每个汽车对象中，其他属性的值都各不相同，但所有对象的方法都相同。

如果你认为这挺好，是在重用代码，请不要这么快下结论。我们确实是在重用这些代码，但这种重用是通过复制实现的：不是只编写一次，而是编写了很多次！如果要改变drive方法的工作方式，结果将如何呢？必须重新编写每个汽车对象的drive方法。这不好，不仅浪费时间，还容易出错。

还有一个比简单复制粘贴更严重的问题：我们假定给对象添加前述所有属性后，就能使其成为汽车对象。如果不小心在对象中遗漏了属性mileage，它还是汽车对象吗？

这些都是前述代码存在的问题，我们将在稍后的一章中处理这些问题，届时将讨论一些在对象中妥善重用代码的技巧。



办法之一是迭代这个对象的属性。为此，可使用还未介绍过的一种迭代方式`for in`，这种迭代器以随机方式遍历对象的所有属性。下面演示了如何显示对象`chevy`的所有属性。

for in以每次一个的方式遍历对象的属性，并依次将每个属性赋给变量prop。

```
for (var prop in chevy) {
    console.log(prop + ": " + chevy[prop]);
}
```

你可使用方括号表示法通过prop来访问当前属性。

这指出了另外一点：还有一种访问属性的方式。刚才访问对象`chevy`的属性时，我们使用了另一种语法，你注意到了吗？实际上，访问对象的属性时，有两种方式可供选择。一种是你已经知道的句点表示法：

```
chevy.color
```

使用对象名、句点和属性名。

JavaScript控制台

```
make: Chevy
model: Bel Air
year: 1957
color: red
passengers: 2
convertible: false
mileage: 1021
```

另外一种方式是方括号表示法，类似于下面这样：

```
chevy["color"]
```

使用对象名以及用引号和方括号括起的属性名。

与访问数组元素的方式有点像。

你需要知道的是，这两种方式是等价的，功能相同。唯一的差别在于，在有些情况下，方括号表示法更灵活些，因为可以像下面这样使用表达式来指定属性名：

```
chevy["co" + "lor"]
```

可将任何表达式放在方括号内，只要其结果为表示属性名的字符串即可。

行为如何影响状态
给车加点油



对象包含**状态**和**行为**。对象的属性让你能够记录对象的状态，如油位、当前温度或收音机当前播放的歌曲。对象的方法让你能够实现行为，如发动汽车、加热或快进。状态和行为会**相互影响**，你注意到了吗？例如，没有油就无法发动汽车，而开车时油量会逐渐减少。有点像现实生活，不是吗？

下面来进一步实现这种概念，给车加点油，然后就可以添加有趣的行为了。要给车加油，需要添加新属性`fuel`和新方法`addFuel`。方法`addFuel`有一个形参`amount`，我们使用它来增加属性`fuel`存储的油量。下面来给对象`fiat`添加这些属性：

```
var fiat = {
    make: "Fiat",
    model: "500",
    // 其他属性。为节省篇幅，省略了它们
    started: false,
    fuel: 0,

    start: function() {
        this.started = true;
    },
    stop: function() {
        this.started = false;
    },
    drive: function() {
       if (this.started) {
         alert(this.make + " " + this.model + " goes zoom zoom!");
       } else {
         alert("You need to start the engine first.");
       }
    },
    addFuel: function(amount) {
        this.fuel = this.fuel + amount;
    }
};
```

我们添加了新属性fuel，用于存储汽车的油量。一开始，这辆汽车的油箱是空的。

我们还添加了方法addFuel，用于给汽车加油。想加多少就可以加多少，只需在调用这个方法时指定即可。

别忘了，fuel是一个属性，因此需要使用关键字this。

而amount是一个函数形参，因此不需要使用关键字this。

图灵社区会员 Kensuke(cpy4ever@gmail.com) 专享 尊重版权