HZ BOOKS
华章科技

世界级JavaScript程序员力作，JavaScript之父Brendan Eich高度评价并强力推荐。

JavaScript编程原理与运用规则完美融合，读者将在游戏式开发中学会JavaScript程序设计，是系统学习JavaScript程序设计的首选之作。

华章程序员书库

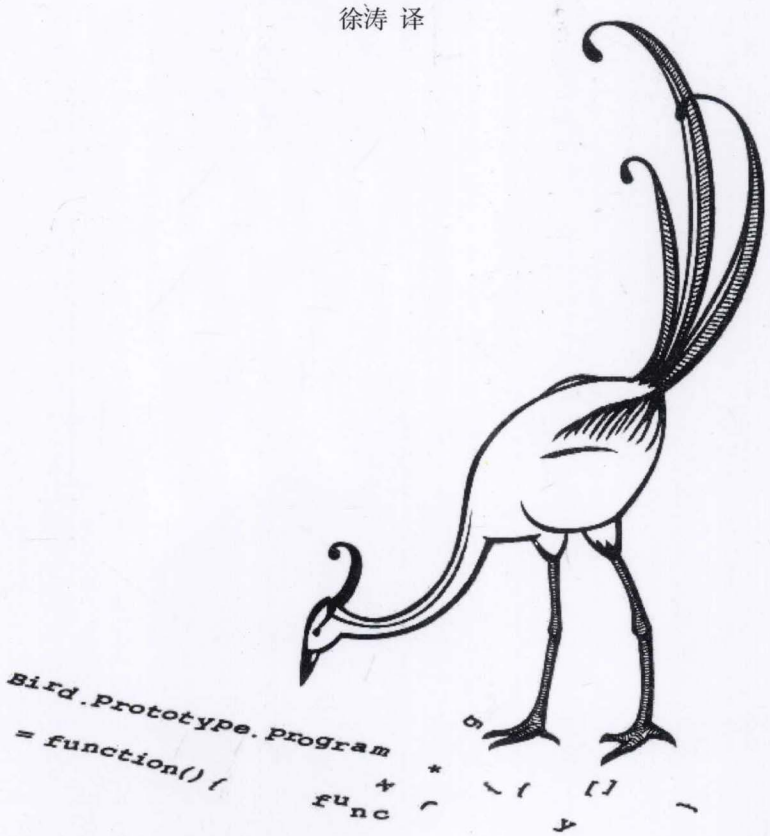
Eloquent JavaScript

A Modern Introduction to Programming

JavaScript编程精解

（美） Marijn Haverbeke 著

徐涛 译



机械工业出版社
China Machine Press

本书仅提供部分阅读，如需完整版，请联系QQ: 461573687

提供各种书籍pdf下载，如有需要，请联系 QQ: 461573687

PDF制作说明:

本人可以提供各种PDF电子书资料，计算机类，文学，艺术，设计，医学，理学，经济，金融，等等。质量都很清晰，而且每本100%都带书签和目录，方便读者阅读观看，只要您提供给我书的相关信息，一般我都能找到，如果您有需求，请联系我 QQ: 461573687, 或者 QQ: 2404062482。

本人已经帮助了上万人找到了他们需要的PDF，其实网上有很多PDF,大家如果在网上不到的话，可以联系我QQ。因PDF电子书都有版权，请不要随意传播，最近pdf也越来越难做了，希望大家尊重下个人劳动，谢谢!

备用QQ:2404062482

编程原理与运用规则的简练、完美融合。我喜欢游戏式的程序开发教程，本书再次点燃了我学习编程的热情。对了，是JavaScript！

—— **Brendan Eich** JavaScript之父

因为这本书，我成为了更棒的架构师、作家、咨询师和开发者。

—— **Angus Croll** Twitter开发者

如果你决定只买一本有关JavaScript的书，那么就应是Marijn Haverbeke的这本书。

—— **Joeyde Villa** GlobalNerdy

本书不仅是学习JavaScript最棒的教材之一，而且是通过学习JavaScript进而学习现代编程的优秀图书。当有人问我如何学好JavaScript时，我会推荐这本书。

—— **Chris Williams** 美国JSConf的组织者

我读过的最棒的JavaScript书籍之一。

—— **Rey Bango** 微软Client-Web社区项目经理和jQuery团队成员

这本书不仅是一本非常不错的JavaScript指导书，而且是一本很棒的编程指导书。

—— **Ben Nadel** Epicenter咨询公司首席软件工程师

这本书对编程基本原理的详述以及对栈和环境等概念的解释非常到位。注重细节使本书从其他的JavaScript书中脱颖而出。

—— **Designora**

上架指导：计算机/程序设计/Web开发
ISBN 978-7-111-39665-9
9 787111 396659
定价：49.00元

客服热线：(010) 88378991, 88361066
购书热线：(010) 68326294, 88379649, 68995259
投稿热线：(010) 88379525
读者信箱：hzjsj@hzbook.com

华章网站 http://www.hzbook.com
网上购书：www.china-pub.com

华章程序员书库

Eloquent JavaScript
A Modern Introduction to Programming

JavaScript编程精解

（美） Marijn Haverbeke 著
徐涛 译

机械工业出版社
China Machine Press

如果你只想阅读一本关于 JavaScript 的图书，那么本书应该是你的首选。本书由世界级 JavaScript 程序员撰写，JavaScript 之父和多位 JavaScript 专家鼎力推荐。本书适合作为系统学习 JavaScript 的参考书，它在写作思路上几乎与现有的所有同类书都不同，打破常规，将编程原理与运用规则完美地结合在一起，而且将所有知识点与一个又一个经典的编程故事融合在一起，读者可以在轻松的游戏式开发中学会 JavaScript 程序设计，趣味性十足，可操作性极强。

全书一共 12 章：第 1~3 章介绍了 JavaScript 的基本语法，旨在帮助读者编写出正确的 JavaScript 程序，包含数字、算术、字符串、变量、程序结构、控制流程、类型、函数、对象和数组等最基础和最核心的内容；第 4~7 章讲解了 JavaScript 编程中的高级技术，目的是帮助读者编写更复杂的 JavaScript 程序，主要涉及错误处理、函数式编程、面向对象编程、模块化等重要内容；第 8~12 章则将重心转移到 JavaScript 环境中可用的工具上，分别详细讲解了正则表达式、与 Web 编程相关的知识、文档对象模型、浏览器事件和 HTTP 请求等。

Copyright © 2011 by Marijn Haverbeke. Title of English-language original: Eloquent JavaScript, ISBN 978-1-59327-282-1, published by No Starch Press.

Simplified Chinese-language edition copyright © 2012 by China Machine Press.

No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or any information storage and retrieval system, without permission, in writing, from the publisher.

All rights reserved.

本书中文简体字版由 No Starch Press 授权机械工业出版社在全球独家出版发行。未经出版者书面许可，不得以任何方式抄袭、复制或节录本书中的任何部分。

封底无防伪标均为盗版

版权所有，侵权必究

本书法律顾问　北京市展达律师事务所

本书版权登记号：图字：01-2012-2098

图书在版编目（CIP）数据

JavaScript 编程精解 /（美）哈弗贝克（Haverbeke，M.）著；徐涛译 . —北京：机械工业出版社，2012.9
（华章专业开发者丛书）
书名原文：Eloquent JavaScript: A Modern Introduction to Programming

ISBN 978-7-111-39665-9

Ⅰ. J…　Ⅱ. ①哈…　②徐…　Ⅲ. JAVA 语言－程序设计　Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2012）第 210644 号

机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码　100037）
责任编辑：秦　健
北京市荣盛彩色印刷有限公司印刷
2012 年 10 月第 1 版第 1 次印刷
186mm×240mm • 11 印张
标准书号：ISBN 978-7-111-39665-9
定价：49.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991；88361066
购书热线：（010）68326294；88379649；68995259
投稿热线：（010）88379604
读者信箱：hzjsj@hzbook.com

对本书的赞誉

编程原理与运用规则的简练、完美融合。我喜欢游戏式的程序开发教程。本书再次点燃了我学习编程的热情。对了，是 JavaScript！

——Brendan Eich，JavaScript 之父

因为这本书，我成为了更棒的架构师、作家、咨询师和开发者。

——Angus Croll，Twitter 开发者

如果你决定只买一本有关 JavaScript 的书，那么就应是 Marijn Haverbeke 的这本书。

——Joey deVilla，Global Nerdy

本书不仅是学习 JavaScript 最棒的教材之一，也是通过学习 JavaScript 进而学习现代编程的优秀图书。当有人问我如何学好 JavaScript 时，我会推荐这本书。

——Chris Williams，美国 JSConf 组织者

我读过的最棒的 JavaScript 书籍之一。

——Rey Bango，微软 Client-Web 社区项目经理和 jQuery 团队成员

这本书不仅是一本非常不错的 JavaScript 指导书，而且是一本很棒的编程指导书。

——Ben Nadel，Epicenter 咨询公司首席软件工程师

真是本好书。

——Design Shack

这本书对编程基本原理的详述以及对栈和环境等概念的解释非常到位。注重细节使本书从其他的 JavaScript 书中脱颖而出。

——Designorati

学习 JavaScript 的好书。

——Craig Buckler，OptimalWorks Web Design 公司

译 者 序

当我第一次阅读这本书的时候，就深深地喜欢上了这本书的写作风格。游戏式的章节、完整且连贯的故事，这些都使我在阅读过程中真正有了读书的快感。

与其他的 JavaScript 书籍不同，本书没有列表式的数据类型讲解，也没有枯燥的概念和老掉牙的例子，更没有流行的 Ajax 专题。本书通过设计一个个由浅入深的小游戏，让读者更加深入而轻松地学习如何应用 JavaScript 编程技术。因此，建议读者在阅读过程中，每次阅读一个完整章节，以便更好地理解编程故事的情节。

在翻译过程中，除了对 JavaScript 语言本身有了深刻理解之外，我也学到了如何从一个需求（游戏设计）进行分析，进而细化到可编程的 JavaScript 代码部分，尤其是本书中虚拟生态圈游戏的设计，使我真正体会到了 JavaScript 原来还可以这么做。

我非常荣幸能够参与本书的翻译工作，感谢机械工业出版社的编辑在翻译过程中给予的鼓励和信任。与此同时，也要感谢我的家人在翻译过程中对我的支持和理解，尤其是我的爱人对本书进行了无数次的阅读，并给出了大量的修改建议。

由于译者水平有限，翻译错误或者风格不合口味在所难免，对你造成的阅读上的不便我深表歉意。

针对本书的任何意见你都可以在我的博客上（http://www.cnblogs.com/TomXu），或者通过邮件直接发给我，我的邮件地址是 taoxu@live.com。

感谢并期待你的批评和指正！

前　言

20世纪70年代，业界首次推出个人计算机时，大多数计算机都内置一种简单的编程语言——通常是BASIC的变体，人与计算机之间的互动需要通过这种语言实现。这意味着，对天生喜欢钻研技术的人来说，从单纯使用计算机到编程的过渡非常容易。

现在的计算机相比20世纪70年代的功能更加强大，价格也更加便宜，软件接口呈现的是使用鼠标操作的灵活图形界面，而不是语言界面。这使计算机更容易使用，总的来说，这是一个巨大的进步。然而，这也在计算机用户与编程世界之间制造了一个障碍——业余爱好者必须积极寻找自己的编程环境，而不是一打开电脑就呈现的环境。

实质上，计算机系统仍然被各种编程语言控制。大多数的编程语言都比早期个人计算机中的BASIC语言更加先进。例如，本书的主题——JavaScript语言，就存在于每一款主流Web浏览器中。

关于编程

不愤不启，不悱不发。举一隅不以三隅反，则不复也。

——孔子

本书除了介绍JavaScript外，也致力于介绍编程的基本原理。事实上，这种编程还是比较难的。编程的基本规则通常都简单明了，但计算机程序构建在这些基本规则之上后，会变得很复杂，产生了其自身的规则和复杂性。正因为如此，编程并不是那么简单或可预测的。正如计算机科学的鼻祖高德纳（Donald Knuth）所说，编程是一门**艺术**，而不是一门科学。

要想从本书里获取最大收获，不能仅仅依靠被动阅读。一定要集中注意力去理解示例代码，只有确定自己真正理解了前面的内容，才能继续往下阅读。

程序员对其创造的宇宙负全部责任，因为他们是创造者。以计算机程序的形式，可创造出无限复杂的宇宙。

——Joseph Weizenbaum，《Computer Power and Human Reason》

一个程序包含很多含义。它是程序员敲出的一串字符，是计算机运行的指向力，是计算机内存中的数据，还控制同一个内存上的执行动作。仅使用熟悉的类推法比较程序与对象往往还不够，因为从表面上看适合该操作的是机器。机械表的齿轮巧妙地啮合在一起，如果表的制造者技术很棒，它就能够连续多年准确地显示时间。计算机程序的元素也以类似的方式组合在一起，如果程序员知道自己在做什么，那么这个程序就能够正常运行而不会崩溃。

计算机作为这些无形机器的载体而存在。计算机本身只会做简单直接的工作。它们之所以

如此有用，是因为它们能够以惊人的速度完成这些工作。程序可以巧妙地把许多简单动作结合起来，去完成非常复杂的工作。

对有些人来说，编写计算机程序是一种很有趣的游戏。程序是思想的构筑，它零成本、零重量，在我们的敲打中不断发展。如果我们不细心，它的规模和复杂性将失去控制，甚至创造者也会感到混乱。这就是编程的主要问题：控制好程序。程序在工作时是很不可思议的，编程的艺术就是控制复杂性的技巧，好的程序其复杂性也会降低。

如今，很多程序员认为只要在程序中使用少量易于理解的技术，就可以最有效地降低复杂性。他们制定了严格的编程规则（**最佳实践**）及书写格式，那些破坏规则的人被称为“差劲”的程序员。

丰富多彩的编程世界里包含了太多的复杂性！让我们努力将程序变得简单和可预测，并为所有奇妙和优美的程序制定禁忌规则。编程技术的前景是广阔的，其多样性使人着迷，它的世界仍有很多未被探索的部分。编程过程中有很多陷阱和圈套，缺乏经验的程序员会犯各类糟糕的错误，告诫我们需要谨慎，并保持头脑清醒。学习编程时总是需要探索新的挑战、新的领域，拒绝不断探索的程序员必定会停滞不前、忘记编程的快乐、并失去编程的意志（或成为管理人员）。

语言为何很重要

在计算机诞生初期并没有编程语言。程序看起来就像这样：

```
00110001 00000000 00000000
00110001 00000001 00000001
00110011 00000001 00000010
01010001 00001011 00000010
00100010 00000010 00001000
01000011 00000001 00000000
01000001 00000001 00000001
00010000 00000010 00000000
01100010 00000000 00000000
```

这是一个从 1 加到 10 并输出结果（1 + 2 +…+ 10 = 55）的程序。它可以在一个非常简单、理想化的计算机上运行。为早期的计算机编制程序时，必须在正确的位置设置一排排的开关或者在纸带上打上一系列有规律的孔点，这样才能将程序传递给计算机。可以想象这个过程有多么繁琐和易出错。即使编写简单的程序也需要使用很多脑力和规则，编写复杂的程序更是不可想象。

当然，手动输入这些二进制位（即以上这些 1 和 0 的统称）的神秘组合，让程序员感觉自己像巫师一样拥有强大的魔力，而且还能够获得工作满足感，因此这点还是很值得的。

程序的每一行都包含一条单独的指令。可以用语言这样描述：

1）将数字 0 保存在第 0 个存储单元；

2）将数字 1 保存在第 1 个存储单元；

3）将第 1 个存储单元的值保存在第 2 个存储单元；

4）将第 2 个存储单元中的值减去数字 11；

5）如果第 2 个存储单元中的值是数字 0，则继续执行指令 9；

6）将第 1 个存储单元的值添加至第 0 个存储单元；

7）将数字 1 添加至第 1 个存储单元；

8）继续执行指令 3；

9）输出第 0 个存储单元的值。

虽然这比二进制位易读，但仍然令人不快。用名称代替指令和存储单元的数字或许更有帮助。

```
 Set 'total' to 0
 Set 'count' to 1
[loop]
 Set 'compare' to 'count'
 Subtract 11 from 'compare'
 If 'compare' is zero, continue at [end]
 Add 'count' to 'total'
 Add 1 to 'count'
 Continue at [loop]
[end]
 Output 'total'
```

在这里不难看出程序是如何运行的。前两行代码为两个存储单元赋予初始值：total 用于创建计算的结果，count 记录当前看到的数字。使用 compare 的行可能是最令人费解的地方。该程序的目的是判断 count 是否等于 11，从而确定能否停止运行。由于该机器相当原始，它只能测试一个数字是否为零，并在此基础上做出判断（跳转），因此它使用标记 compare 的存储单元来计算 count 的值，即 11，并在该值的基础上做出判断。后面两行代码将 count 的值添加到结果 total 上，当程序判断 count 不是 11 时，为 count 加 1。

下面是用 JavaScript 编写的有同样效果的程序。

```
var total = 0, count = 1;
while (count <= 10) {
  total += count;
  count += 1;
}
print(total);
```

这段程序有了更多的改进。最重要的是，不再需要指定程序来回转换的方式，while 这个神奇的单词会帮助解决这个问题。只要满足给定的条件：count <= 10（意思是“count 小于或等于 10”），它就会继续执行下面几行代码。不再需要创建临时的值并将该值与零比较——这是一个没有意义的细节，编程语言就是用于解决这些无意义的细节。

最后，如果我们可以使用方便的 range 和 sum 操作（分别在区间范围（range）里创建一组数字，并计算该组数字的总和（sum）），代码应该是这样的：

```
print(sum(range(1, 10)));
```

从上例可以看出，同样的程序可以使用长或短、不可读或可读的方式来表达。该程序的第一个版本非常晦涩，而最后一个版本基本上都是语言描述：打印（print）出从 1 到 10 这个区间范围（range）的总和（sum）。（我们将在后面的章节中讲述如何创建 sum 及 range 等函数。）

优秀的编程语言会提供一种更为抽象的表达方式来帮助程序员表达其意图。这种语言隐藏无意义的细节，提供方便的程序块（例如 while 语句），在很多时候，它允许程序员自己添加程序块（例如 sum 和 range 操作）。

什么是 JavaScript

JavaScript 语言目前主要用于解决万维网页面的各种难题。近年来，该语言也开始应用于其他环境中，如 node.js 框架（一种使用 JavaScript 语言编写快速服务器端程序的方式），最近吸引了不少关注。如果对编程感兴趣，JavaScript 绝对是值得学习的语言。即使以后不会从事大量的网络编程工作，本书中展示的一些程序也会一直伴随着你，影响你使用其他语言编写的程序。

有些人指出了 JavaScript 语言不好的地方，其中很多观点都是对的。当我第一次用 JavaScript 写程序时，我立刻就开始轻视这种语言了，它接受我输入的大部分代码，但其解释代码的方式与我的意图完全不同。无可否认，这与我做事没有头绪有很大的关系，但也存在着一个真正的问题：JavaScript 允许的操作实在是太多了。这种设计的初衷是让初学者更易掌握 JavaScript 编程方法。实际上，这使得更难发现程序中的问题，因为系统不会指出问题所在。

然而，语言的灵活性也是一种优势。它为很多无法使用更强大语言的技术留下了发展空间，正如我们将在后面几章中看到的，它能够克服 JavaScript 的一些缺点。当正确学习 JavaScript 并使用了一段时间之后，我发现自己真正的喜欢上了这种语言。

与这种语言的名字所暗示的不同，JavaScript 与 Java 编程语言并没有多大关系。相似的名字是基于营销的考虑，而不是为了获得好评。网景公司于 1995 年推出 JavaScript 时，Java 语言正在极力推广中，并且很受欢迎。很显然，当时有人认为借助 Java 语言的成功进行营销是个好主意。于是，就有了现在看到的这个名字。

与 JavaScript 有关的是 ECMAScript。当网景浏览器以外的浏览器开始支持 JavaScript 或类似语言时，出现了一份准确描述该语言应该如何工作的文档。此文档里所描述的语言称为 ECMAScript，它是以制定该设计语言标准的欧洲计算机制造商协会（ECMA）的名称命名的。ECMAScript 描述了一种多用途编程语言，但并没有提及它与 Web 浏览器的集成。

现在有多个版本的 JavaScript。本书讲的是 ECMAScript 3 版本，是第一个各种浏览器都支持的版本。在过去的几年中，人们进一步发展这种语言，但是，这些扩展版本只有受到浏览器的广泛支持才是有用的（至少在网络编程方面），而浏览器要跟上编程语言的发展也需要很长一段时间。幸运的是，新版本的 JavaScript 基本上是 ECMAScript 3 的扩展版本，因此本书中的大部分内容都不会过时。

试运行程序

如果想要执行并练习本书中的代码，一种方法是可以登录 http://eloquentjavascript.net/，使用该网站提供的工具。

另一种方法是创建一个包含该程序的 HTML 文件，并将其加载到浏览器中。例如，可以创建一个名为 test.html 的文件，内容如下。

```
<html><body><script type="text/javascript">

var total = 0, count = 1;
while (count <= 10) {
  total += count;
  count += 1;
}
document.write(total);

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

后面的章节将讲述更多有关 HTML 的知识以及浏览器解释 HTML 的方式。请注意，示例中的 print 操作已被替换为 document.write。我们将在第 10 章中介绍如何创建 print 函数。

本书主要内容

本书前 3 章介绍了 JavaScript 语言，并指导如何编写语法正确的 JavaScript 程序，还介绍了控制结构（如在前面看到的 while 关键词）、函数（自己编写的操作）和数据结构。这些内容可以指导编写简单的程序。

在基本了解了编程的基础上，后面 4 章将讨论更高级的技术，指导如何编写更复杂的程序，而不是将程序写得难以理解。第 4 章将讨论如何处理错误和意想不到的情况。第 5 章和第 6 章将介绍两个主要的抽象方式：函数式编程和面向对象编程。第 7 章给出了一些如何组织程序的指导。

其余的章节重点关注了 JavaScript 环境中可用的工具，理论部分相对较少。第 8 章介绍了一种文字处理的子语言，第 9~12 章将讲述程序在浏览器内部运行时可利用的工具，教会大家控制网页、应对用户的操作，并与 Web 服务器通信。

排版规范

在本书中，采用等宽字体的文本应被理解为代表程序的元素，这些元素有时是各自独立的片段，有时只是指靠近程序的一部分。这段我们已经看过的程序的一部分，可写成如下形式：

```
function fac(n) {
  return n == 0 ? 1 : n * fac(n - 1);
}
```

有时，为了演示某些表达式求值时会发生的情况，这些表达式会采用黑体字，其产生的值写

X

在下面，前面有一个箭头：

```
1 + 1
→ 2
```

更新

访问 http://nostarch.com/ejs.htm 获得更新、勘误表以及运行本书示例代码的交互式代码沙盒。

目　录

第①章

JavaScript 基础：值、变量、控制流程

计算机世界里只有数据，没有数据计算机就不存在。所有数据实质上都是由 bit 序列构成的，因此基本上都是相似的。bit 序列通常是由 0 和 1 两种数字排列组合而成的，它们在计算机内的形式就如一个高电荷或一个低电荷、一个强信号或一个弱信号，或光盘表面的一个亮点或一个暗点。

1.1 值

虽然构成相同，但每一部分数据都扮演着自己的角色。在 JavaScript 系统中，大多数数据都被有序地分成了各种值。每个值都有一个类型，用于确定它扮演的角色类型。JavaScript 里有 6 种基本类型的值：number、string、Boolean、object、function 和 undefined。

创建值的时候，只需调用它的名称即可，非常方便。无需为创建的值收集构建素材或是支付费用，只要调用某个值，便可立即获得该值。当然，值也不是凭空创建的，每个值都需要存储在某个地方，如果在同一时间使用大量的值，就有可能耗尽内存，幸运的是，只有在同时使用大量数据的时候才会出现这个问题。一旦不再需要这个值，它将会消失，只剩下一些 bit 数据，用于再次生成值。

1.1.1 数字

number 类型的值就是数字值，它们通常写成如下这样的数字：

```
144
```

将 144 输入程序，144 就会在计算机内部存储起来。在 bit 里如下面所示存放 144：

```
01000000011000100000000000000000000000000000000000000000000000000000
```

如果读者认为像 10010000（144 的整数表示法）这样的存放方式还不错，在某些情况下可能确实需要采用这样的表示方法，但标准的 JavaScript 数字描述是 64 位的浮点型值。因此，这些值也可以包括分数和指数。

但是，本书不会深入研究二进制表示法，我们更感兴趣的是这种表示法对数字产生的实际影响。首先，数字表示实际上是有 bit 数量限制的，也就是有精度限制的。64 个 1 或 0 的值只能表

示 2^{64} 个数字。这个数字已经很大了，超过了 10^{19}。

能够在 JavaScript 中使用的数字并非所有小于 10^{19} 的正整数，还包括负数，因此需要使用其中一个 bit 来存储数字符号。更大的难题在于：必须将非整数表示出来。为此，还需要使用 11 个 bit 来存储数字的小数点位置。

这样就剩余 52 个 bit[⊖]，任何小于 2^{52} 的整数（大于 10^{15}）都可以安全地写成 JavaScript 数字。大多数情况下使用的数字是小于该数值的。

使用点（.）来写入小数：

```
9.81
```

对于任何非常大或者非常小的数字，可以添加一个 e，用科学计数法来表示该数值，e 后面跟的是数字的指数。

```
2.998e8
```

这里表示 2.998×10^8=299800000。

用于整数计算时整数（integer）存放在 52 个 bit 内，计算结果通常十分精确，但小数计算的精确度一般不高。例如 π 无法通过有限的小数数字来精确表示，当只有 64 个 bit 用于存放数字时，很多数字的精度就会降低。这是件很糟糕的事情，不过只有在极特殊情况下才会出现这样的问题。了解这一点很重要，因此应将小数视为近似值而不是精确值。

1.1.2 算术

与数字密切相关的就是算术，加法或乘法等算术运算需要使用两个数字，并利用它们产生一个新数字。JavaScript 的算术运算如下所示：

```
100 + 4 * 11
```

符号“+”和“*”被称为**运算符**（操作符），第一个符号代表加法，第二个符号代表乘法。在两个数字之间加上运算符后，这两个数字将应用该运算符计算出一个新数字。

上面的示例是说“4 加 100 以后将结果和 11 相乘”，还是在加法之前先做乘法？正如我们猜到的，先做乘法。但在数学运算上，可以用括号把加法括起来改变运算顺序。

```
(100 + 4) * 11
```

减法使用“−”运算符，除法使用“/”运算符。如果出现多个运算符，而又没有括号，运算的顺序是按照运算符的优先级来确定的。在上面第一个示例中，乘法的优先级高于加法。除法的优先级与乘法的优先级相同，加减法的优先级也相同。如果同时出现多个同优先级的运算符（如 1−2 + 1），应按照从左到右的顺序运算。

不需过多考虑这些优先级规则，如果不确定时，就使用括号。

有一个运算符可能不太熟悉，“%”符号表示**余数**，X%Y 表示 X 除以 Y 的余数。例如，314 % 100 的结果是 14，10 % 3 的结果是 1，144 % 12 的结果是 0。% 运算符的优先级与乘除法相同。

⊖ 实际上是 53 个 bit，因为有一个方法可以免费获取一个 bit，详情可以查看 IEEE 754 格式。

1.1.3 字符串

另外一个数据类型是字符串（string），它不像数字那样从名称就能明显看出其用途，但也起到非常基础的作用。string 用于表示文本（其名称可能是因它能够串起一堆字符而得的）。string 的书写方式是用引号将内容括起来。

```
"Patch my boat with chewing gum."
'You ain\'t never seen a donkey fly!'
```

单引号和双引号均可以用来标记 string，只要引号前后一致就可以。

几乎任何字符都可以放在引号里，JavaScript 会将它解析成 string。但有些字符放在引号里就比较复杂，例如，在引号里放引号就是很有难度的。按下回车键产生的新行也不能放在引号里，string 只能放在一行里。

要将这些特殊字符放在 string 里，需要遵守下列规则：当在加引号的文字里发现反斜杠（\），那就意味着其后面的字符有特殊意义。反斜杠后加引号（\"）并不意味着字符串的结束，而是字符串的一部分。反斜杠后面加 n（\n）表示一个新行，反斜杠后面加 t（\t）表示一个 tab 字符。可以参考下面的示例。

```
"This is the first line\nAnd this is the second"
```

其中包含的真正文字是：

```
This is the first line
And this is the second
```

还有一些情况就是，我们希望字符串的反斜杠只是代表反斜杠，而不是一个特殊的代码。如果两个连续的“\”出现在字符串里，输出的结果里只会有一个“\”。如下就是字符串 A newline character is written like"\n" 的写法：

```
"A newline character is written like \"\\n\"."
```

字符串不能相除、相乘或相减，但可以使用“+”运算符，“+”不表示相加，而表示连接——将两个字符串连接在一起。下面的代码行将产生字符串“concatenate”。

```
"con" + "cat" + "e" + "nate"
```

还有更多操作字符串的方式，稍后讨论。

1.1.4 一元操作符

并不是所有的运算符都是符号，有些运算符是单词。例如 typeof 运算符，它产生一个字符串值，该字符串命名给定值的类型。

```
typeof 4.5
→ "number"
typeof "x"
→ "string"
```

其他运算符都是对两个值执行运算，而 typeof 只对一个值执行运算。使用两个值的运算符称为**二元运算符**，而使用一个值的运算符则称为**一元运算符**。减法“-”运算符可同时用作上述两种运算符。

```
- (10 - 2)
→ -8
```

1.1.5 布尔值、比较和布尔逻辑

接下来了解布尔类型的值，布尔类型只有两个值：true 和 false。以下是产生这两个值的方式。

```
3 > 2
→ true
3 < 2
→ false
```

之前已经看到“>”和“<”符号了，它们的意思分别是：大于和小于。“>”和“<”是二元运算符，应用该运算符的结果是布尔值，表示在此情况下运算符产生的是否为 true。

字符串也可以用同样的方式进行比较。

```
"Aardvark" < "Zoroaster"
→ true
```

字符串基本是按照字母顺序来进行比较的。大写字母始终小于小写字母，所以“Z”<“a”是 true，非字母字符（!、@ 等）也包括在此排序中。比较这些字符的实际方式是基于 Unicode 标准。该标准为每个字符赋予一个数字（包括希腊语、阿拉伯语、日语、泰米尔语等字符）。使用数字对应字符主要用于在计算机中存储字符串——可以将字符表示成一堆数字。比较字符串时，JavaScript 实际上是从左向右逐个比较每个字符所对应的数字代码。

其他类似的运算符还有 >=（大于或等于）、<=（小于或等于）、==（等于）和 !=（不等于）。

```
"Itchy" != "Scratchy"
→ true
```

还有一些运算符可以应用于布尔值本身，JavaScript 支持三种逻辑运算符：与、或、非。它们可以用于推理布尔值。

&& 运算符表示逻辑**与**，它是一个二元运算符，只有在两个值都是 true 的情况下其结果才是 true。

```
true && false
→ false
true && true
→ true
```

|| 运算符是逻辑**或**，如果两个值的任何一个值为 true，其结果就为 true。

```
false || true
→ true
false || false
→ false
```

非运算符用感叹号“!”表示，它是一元运算符，能够反转给定的值。!true 的结果是 false，

而 ! false 的结果是 true。

当布尔运算符与算术以及其他运算符混合使用时，使用括号后的运算顺序并不是很明显。在实践中，了解了目前为止所见到的运算符的用法，就可以理解运算符的意义。|| 的优先级最低，其次是 &&，然后是比较运算符（>、== 等），最后是其他的运算符。在典型的编程情况下，选择运算符时应尽可能少用括号。

1.1.6 表达式与语句

到目前为止，所有示例使用的语言都好像在使用一个袖珍计算器：将一些值通过应用运算符得到新的值。像这样创建值是每个 JavaScript 程序都有的基本内容，但仅仅是一部分。一段创建值的代码称为表达式。每一个直接写出来的值（数字“22”或者字符“psychoanalysis”）都是一个**表达式**，括号之间的表达式也是表达式。将二元运算符应用于两个表达式或者将一元运算符应用于一个表达式，这些也都是表达式。利用这些规则，可以建立任意大小和难度的表达式。（JavaScript 实际上有更多建立表达式的方式，在恰当的时候会揭示这些方式。）

有一个比表达式更大的单位，称为**语句**。一个程序是由一组语句构成的，大多数语句都是以分号（;）结束，最简单的语句是由一个表达式及其后面的分号构成。

```
1;
!false;
```

这是一个没有任何意义的程序。一个表达式可仅用于创建一个值，但一个语句只有发挥改变“世界”的作用，才能算是成功的语句。一个语句可以将某些信息输出到屏幕上（这就是其改变“世界”之举），或者改变程序的内部状态以此来影响后面的语句。这些改变称为**副作用**（side effect），上面示例中的语句仅仅是用于创建两个值——1 和 true，然后立即又将它们扔进 bit 库，一点影响“世界”的痕迹也没有，因此它不是一个副作用。

在某些情况下，JavaScript 允许省略语句后面的分号；而在其他情况下，则必须有分号，否则就会发生异常。关于何时能够安全省略分号的规则十分复杂，其基本思想是：如果一个程序本来是无效的，但是插入分号后就变得有效了，那么该程序能否正常运行就要看它是否有分号。本书中的每个语句后面都用分号结束，强烈建议在程序里也这么做。

1.2 变量

程序如何保持内部状态？又如何存储东西？我们已经了解了如何用旧值产生新值，但这并没有改变旧值，新值必须立即使用，否则就会再次丢失。为了捕获并且拥有这些值，JavaScript 提供了一种功能叫做**变量**。

```
var caught = 5 * 5;
```

变量通常都是有名称的，并且指向一个值（也就是拥有它）。上面的语句表示创建一个叫 caught 的变量，然后指向 5×5 的结果值（也就是 25）上。

声明变量之后，变量的名称可作为一个表达式，用于产生其拥有的值。下面是一个示例。

```
var ten = 10;
ten * ten;
→ 100
```

单词 var 用于创建一个新的变量，var 之后是变量的名称。除了空格之外，几乎任何单词都可以作为变量名。数字也可以作为变量名的一部分（catch22 就是一个合法的名称），但变量不能以数字开头。字符“$”和“_”也可作为字母用于变量名中，因此 $_$ 也是一个合法的变量名。

如果想在声明一个变量之后立即获取一个值（大部分情况都是这样的），可以使用“=”操作符将某些表达式的值赋给这个变量。

当一个变量指向一个值时，并不意味着它永远都是这个值。“=”操作符在任何时候都可以将现有的变量指向新值。

```
caught;
→ 25
caught = 4 * 4;
caught;
→ 16;
```

应该将变量想象成触手而不是盒子，它不“容纳”值，而是“抓取”值——两个变量可以引用同一个值。只有受程序支配的值才能被变量访问。当需要记住某些值的时候，就需要多出一支触手来抓取该值，或者在原有触手的基础上重新接上一支来抓取新的值。

例如，要记住 Luigi 还欠你多少美元，为此创建一个变量，当他还你 35 美元的时候，为该变量赋一个新值。

```
var luigisDebt = 140;
luigisDebt = luigisDebt - 35;
luigisDebt;
→ 105
```

关键字与保留字

请注意，有些名称有特殊的意义，比如 var 不能用作变量名。这些名称称为**关键字**。还有一些单词是在 JavaScript 未来版本里使用的“保留字”。还有一些官方规定的不允许声明为变量名的关键字（尽管某些浏览器允许其运行），这张列表相当长：

```
abstract boolean break byte case catch char class const continue debugger
default delete do double else enum export extends false final finally float
for function goto if implements import in instanceof int interface long native
new null package private protected public return short static super switch
synchronized this throw throws transient true try typeof var void volatile
while with
```

不用记住这些关键字，但要记住如果某些代码不能按预期运行，问题可能就出在关键字上。char（保存一个字符的字符串）和 class 是最常见的，只在 JavaScript 中偶尔使用。

1.3 环境

在给定时间存在的变量和变量值的集合叫做**环境**。当程序开始运行的时候，环境并不是空的，它通常包含一些标准的变量。当浏览器加载一个页面时，它创建一个新环境，并将这些标准的变量存于新环境中。在当前页面中，程序创建和修改的变量直到浏览器打开新页面才会消失。

1.3.1 函数

标准环境提供的很多值都是**函数**类型的，函数是包含在值中的一段程序。通常情况下，这段程序会执行一些有用的操作，使用包含的函数值调用这段程序。在浏览器环境中，变量 alert 拥有一个函数，用于弹出带有消息的小对话框，使用方法如下：

```
alert("Good morning!");
```

执行函数里的代码称为调用或应用，完成这一过程所使用的符号是括号。生成函数值的每个表达式都可以通过在后面添加括号来执行，尽管通常会直接引用包含该函数的变量。在上述代码中，括号中的字符串传递给了 alert 函数，作为文本用于显示在对话框上。给函数传递的值称为**形参**或**实参**，alert 只需要其中一个参数，但其他函数则有可能需要不同数量或不同形式的参数。

弹出对话框是一个副作用，很多函数由于产生了副作用而变得非常有用。另外，函数也可以产生值，这种情况下它就不需要副作用了。例如，函数 Math.max 有两个参数，它返回两个给定数值中较大的值。

```
Math.max(2, 4);
→ 4
```

当函数产生一个值的时候也称为**返回**一个值。因为使用 JavaScript 时，值都是由表达式产生的，所以函数调用也可以作为较长表达式的一部分。

```
Math.min(2, 4) + 100;
→ 102
```

第 2 章将讨论如何编写我们自己的函数。

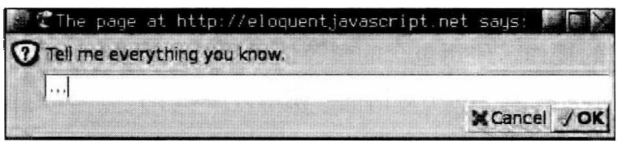
1.3.2 prompt 和 confirm

浏览器提供的标准环境包含了更多用于弹出窗口的函数，可以使用 confirm 函数让用户选择 OK/Cancel 问题。该函数返回布尔值：如果用户单击 OK，则返回 true；如果单击 Cancel，则返回 false。

```
confirm("Shall we, then?");
```

prompt 函数可用于询问一个开放式问题，第一个参数就是该问题；第二个参数是用户需要输入文本的开头部分，可以在对话窗口里输入一行文本，该函数会将其返回作为一个字符串。

```
prompt("Tell me everything you know.", "...");
```



1.3.3 print 函数

如前所述，浏览器提供了 alert 函数用于在小窗口上显示字符串。在这样做测试代码时很有帮助，但每次都单击这些小窗口也会让人感到厌烦，在本书中，假设有一个现成的函数 print，它能够将参数显示在未指定的文本输出设备上，方便编写代码实例。但要注意，Web 浏览器提供的 JavaScript 上下文环境里并不包含该函数。

例如，下面的代码将输入字母 X：

```
print("X");
```

1.3.4 修改环境

在该练习环境中，理论上可以给任何变量赋一个新值，这样做很有用但也很危险。如果给 print 函数赋值为 8，它就不再是一个函数，而且也不能再用于显示信息。第 7 章将讨论如何避免发生重新定义变量的偶然事件。

1.4 程序结构

一行代码的程序意义不大。在程序里编写多行语句时，这些语句都会像预想的一样从上到下逐条执行。下面的程序包含两个语句，第一个语句是让用户输入一个数字，第二个语句显示这个数字的平方值。

```
var theNumber = Number(prompt("Pick a number", ""));
alert("Your number is the square root of " + (theNumber * theNumber));
```

Number 函数将用户输入的值转化成数字（此例中需要使用 Number 函数，是因为 prompt 函数的结果是一个字符串值）。类似的函数还有 String 和 Boolean，它们也都是将值转换成相应的类型。

1.4.1 条件执行

有时并不希望程序里所有的语句都按照同样的顺序执行。例如，在上面的程序中，只想在输入的内容是数字的情况下才计算平方值。

关键字 if 可以根据布尔表达式的值执行或跳过执行语句，可以这样编写：

```
var theNumber = Number(prompt("Pick a number", ""));
if (!isNaN(theNumber))
  alert("Your number is the square root of " + (theNumber * theNumber));
```

if 后面的括号里提供的是条件表达式（本例中是 !isNaN(theNumber)），只有在该表达式的返回值是 true 的情况下，if 后面的语句才会执行。

如果调用 Number 函数传入的参数不是一个数字（如“moo”），则返回的结果是一个特殊值 NaN，用于代替“该结果不是数字”。函数 isNaN 用于判断传入的参数是否为 NaN，因此，如果 theNumber 是数字，则 !isNaN(theNumber) 的值就是 true。

通常情况下，不仅要在条件成立的情况下执行代码，也要在该条件不成立的情况下执行代码。可以将关键字 else 和 if 一起使用，分成两部分执行代码（并联路径）。

```
if (true == false)
  print("How confusing!");
else
  print("True still isn't false.");
```

如果要用多条路径分支，可以多次使用 if/else 对“连”在一起，示例如下。

```
var num = prompt("Pick a number:", "0");

if (num < 10)
  print("Small");
else if (num < 100)
  print("Medium");
else
  print("Large");
```

该程序首先会判断 num 是否小于 10，如果小于 10，则执行输出“small”的分支，然后结束；如果不小于 10，则执行包含第二个 if 的分支。如果第二个条件（小于 100）成立，则表示该数字在 10 和 100 之间，输出“Medium”；如果不成立，则会执行第二个（即最后一个）else 分支。

1.4.2 while 循环与 do 循环

编写一个程序输出 0 到 12 之间所有的偶数。其中一种编写方式如下所示：

```
print(0);
print(2);
print(4);
print(6);
```

```
print(8);
print(10);
print(12);
```

上面的代码是有效的，但编程的目的是**减少**工作，而不是增加工作。如果要打印小于 1000 的所有偶数，上面的代码就不起作用了。编程就是要让一些代码自动重复执行。

```
var currentNumber = 0;
while (currentNumber <= 12) {
  print(currentNumber);
  currentNumber = currentNumber + 2;
}
```

while 开头的语句创建了一个**循环**，循环很像一个条件语句，影响着语句的顺序——它不是不执行语句或仅执行一次语句，而是有可能让这些语句重复执行多次。while 后面的括号里是表达式，用于判断 loop 循环是继续还是结束。只要该表达式产生的布尔值是 true，循环里的代码就会重复执行；如果是 false，程序就会跳到循环的底部，继续执行其他语句。

变量 currentNumber 用于演示变量跟踪程序运行过程的方式。循环每重复一次，变量 currentNumber 的值就会增加 2。每次重复循环开始时，都会和 12 进行比较，判断程序运行是否全部完成。

while 语句的第三部分称为**循环体**，是重复执行的行为代码。如果不使用 print 函数输出数字，程序就如下所示：

```
var currentNumber = 0;
while (currentNumber <= 12)
  currentNumber = currentNumber + 2;
```

此处，currentNumber = currentNumber + 2; 是组成循环体的语句。由于必须输出这个数字，因此循环语句需要有多个，用大括号（{}）将一组语句放到同一个代码**块**中。对于代码块外面的环境来说，这个代码块就相当于一个语句。在本例中，代码块包括 print 输出语句和 currentNumber 更新语句。

编写一个用于计算和显示 2^{10} 的程序作为一个有效的程序示例。可以使用两个变量：一个用于记录结果，另一个用于记录乘了多少次 2。循环判断第二个变量是否达到 10，如果没有达到，则更新这两个变量。

```
var result = 1;
var counter = 0;
while (counter < 10) {
  result = result * 2;
  counter = counter + 1;
}
result;
→ 1024
```

Counter 也是从 1 开始，判断小于等于 10。不过最好是从 0 开始计算，其原因将在后面的内容中解释。

另外一个非常相似的结构是 do 循环。它与 while 循环仅有一点区别：它的循环体至少执行一次，才开始验证是否需要停止循环。下面的循环代码展示了 do 循环的功能。

```
do {
  var input = prompt("Who are you?");
} while (!input);
```

1.4.3 缩进代码

前面的示例中，在一些语句前面输入了空格。这些空格不是必须的——没有空格程序也会正常运行。实际上，即使是程序里的换行符也是可选的，可以将所有的代码都写成一行很长的代码。缩进代码的作用是突出代码结构，代码块里可以有新的代码块，如果在一个非常复杂的程序中不缩进，很难区分某一个代码块是在哪里结束而另一个代码块是从哪里开始。使用代码行缩进以后，程序的外观形状就与其内部代码块的形状相对应。笔者喜欢为每个代码块使用两个空格进行缩进，但每个人的偏好不同，有些人喜欢使用 4 个空格，而有的人则使用 Tab 键。

1.4.4 for 循环

到目前为止，while 循环展示的都是相同的模式。首先，创建 counter 变量用于跟踪循环过程，while 本身包含了一个条件检查用于判断 counter 是否达到边界值。然后，在循环体的底部更新 counter 变量。

很多循环都采用这种模式，JavaScript 和其他类似语言也都提供了更为简洁和易于理解的用法。

```
for (var number = 0; number <= 12; number = number + 2)
  print(number);
```

该程序恰好和前面的输出偶数数字示例是相同的。唯一的变化是，所有与循环的“状态”相关的语句都放在了一行。for 后面的括号里包含两个分号，第一部分通常可以通过**声明**一个变量来初始化循环；第二部分用于**检测**循环是否必须继续；第三部分用于**更新**循环的状态。在大多数情况下，for 语句比 while 语句更加简短和清晰。

下列代码用来计算 2^{10}，用 for 代替 while：

```
var result = 1;
for (var counter = 0; counter < 10; counter = counter + 1)
  result = result * 2;
result;
→ 1024
```

即便代码块没有以大括号“{”开头，循环里的语句前面仍要使用两个空格进行缩进，以便清晰地表示该语句“属于”该循环。

1.4.5 跳出循环

如果一个循环并不总是需要运行到结束，可以使用 break 关键字。break 语句用于立即退出当前循环，继续执行后面的程序。下列程序可用于查找第一个大于 20 并且能被 7 整除的数字。

```
for (var current = 20; ; current++) {
  if (current % 7 == 0)
    break;
}
current;
→ 21
```

百分比（%）操作符是判断一个数字是否可以被另外一个数字整除的最简单的方式：如果可以整除，则余数应该为 0。

本例的 for 结构没有检测是否结束循环的代码，这意味着是否退出循环取决于内部的 break 语句。另外，同样功能的循环代码可以简单编写成如下形式：

```
for (var current = 20; current % 7 != 0; current++)
  ;// 什么都不做
```

在此例中，循环体为空，单独一个分号可以用于产生一个空语句。

1.4.6 更新变量简便法

程序通常需要在原值的基础上为变量更新一个新值，尤其是在循环发生的时候。例如 counter = counter + 1，JavaScript 提供了一种简便方法：counter += 1。其他操作符也可以使用该方法，例如 result *= 2 是计算 result 的 2 倍，而 counter-= 1 则是将 counter 减 1。

对于 counter += 1 和 counter-= 1，还有更简洁的版本：counter++ 和 counter--。

此时，代码会变得更短：

```
var result = 1;
for (var counter = 0; counter < 10; counter++)
  result *= 2;
```

1.4.7 使用 switch 进行调度

下面的代码非常普遍：

```
if (variable == "value1") action1();
else if (variable == "value2") action2();
else if (variable == "value3") action3();
else defaultAction();
```

switch 结构能够以更直接的方式解决这样的“调度”问题，不过，JavaScript 为解决该问题使用的语法（继承自 C 语言和 Java 编程语句）看起来有点笨拙，因此，有时认为 if 语句链仍是较好的选择。下面是一个示例。

```
switch(prompt("What is the weather like?")) {
  case "rainy":
    print("Remember to bring an umbrella.");
    break;
  case "sunny":
    print("Dress lightly.");
  case "cloudy":
    print("Go outside.");
    break;
  default:
    print("Unknown weather type!");
    break;
}
```

代码块内部使用了 switch，可以使用任意数量的 case 标签。程序会根据 switch 给定的值跳转到相应的标签处，如果没有匹配到标签则跳转到默认标签，然后开始执行标签处的语句，紧接着执行其他标签，一直到遇见 break 语句为止。在某些情况下，比如该例中的 sunny 这个 case，可用于在多个 case 里共享代码（无论是 sunny 还是 cloudy 都推荐退出）。但要注意，因为很容易忘记声明 break，所以往往会导致程序执行一些不想执行的代码。

1.4.8 大小写

之所以在声明变量名时采用独特的大小写形式，是因为变量声明不能包含空格。计算机会将带有空格的变量识别成两个单独的变量，如果一个变量名由多个单词组成，只可选用如下几种形式：fuzzylittleturtle、fuzzy_little_turtle、FuzzyLittleTurtle 或者 fuzzyLittleTurtle。第一种形式很难读懂，我个人比较喜欢使用下划线（尽管输入有点麻烦）。标准的 JavaScript 函数和大多数 JavaScript 开发人员都采用最后一种写法，习惯这种小用法不是很难，因此普遍使用第一个单词后面的每个单词的首字母大写。

在少数情况下（比如 Number 函数），变量的第一个字母也是大写。这主要是为了将该函数标记为构造函数。第 6 章将详细讲解什么是构造函数，但现在最重要的是不要被这种明显缺乏一致性的书写形式所牵绊。

1.4.9 注释

在一个程序示例中写了 // Do nothing 这个部分，大家可能会有点疑惑。在程序里添加额外的文本通常是非常有用的，最常见的用法就是给程序添加注释。

```
// The variable counter, which is about to be defined, is going
// to start with a value of 0, which is zero.
var counter = 0;
// Next, we loop. Hold on to your hat.
while (counter < 100 /* counter is less than one hundred */)
```

本书仅提供部分阅读，如需完整版，请联系QQ: 461573687

提供各种书籍pdf下载，如有需要，请联系 QQ: 461573687

PDF制作说明:

本人可以提供各种PDF电子书资料，计算机类，文学，艺术，设计，医学，理学，经济，金融，等等。质量都很清晰，而且每本100%都带书签和目录，方便读者阅读观看，只要您提供给我书的相关信息，一般我都能找到，如果您有需求，请联系我 QQ: 461573687, 或者 QQ: 2404062482。

本人已经帮助了上万人找到了他们需要的PDF，其实网上有很多PDF,大家如果在网上不到的话，可以联系我QQ。因PDF电子书都有版权，请不要随意传播，最近pdf也越来越难做了，希望大家尊重下个人劳动，谢谢!

备用QQ:2404062482