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

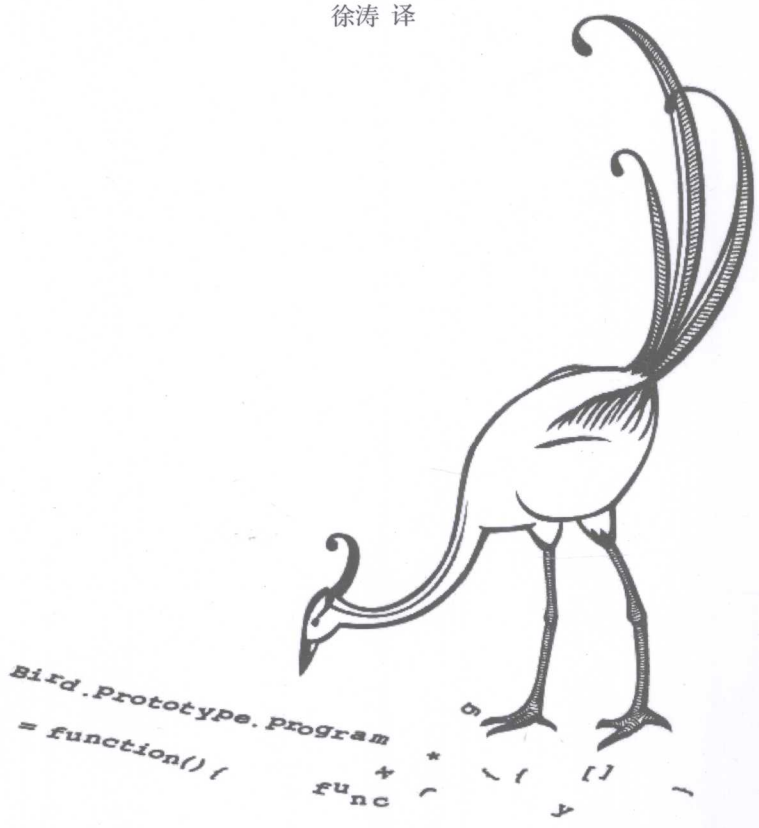
Eloquent JavaScript

A Modern Introduction to Programming

JavaScript编程精解

（美） Marijn Haverbeke 著

徐涛 译



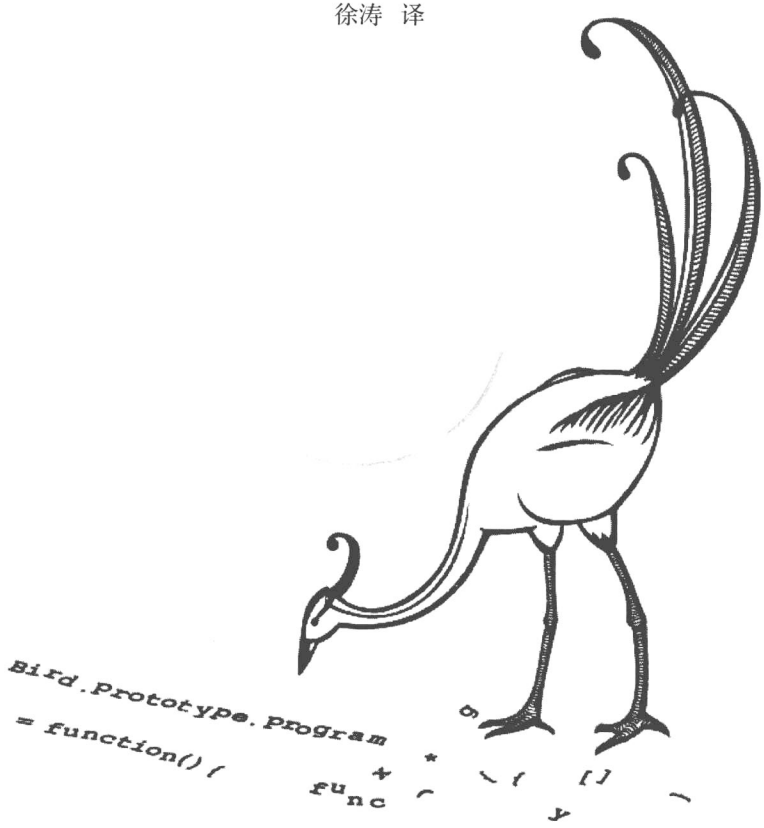
机械工业出版社
China Machine Press

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。

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

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：

```
01000000011000100000000000000000000000000000000000000000000000000
```

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。