Web 开发与设计

深入理解现代 JavaScript

[美] T. J. 克罗德(T. J. Crowder) 著
赵 永 卢贤泼 译

清华大学出版社
北 京

北京市版权局著作权合同登记号　图字：01-2020-6113
All Rights Reserved. This translation published under license. Authorized translation from the English language edition, entitled JavaScript: The New Toys, ISBN 9781119367956, by T. J. Crowder, Published by John Wiley & Sons. Copyright © 2020 by Thomas Scott “T. J.” Crowder. No part of this book may be reproduced in any form without the written permission of the original copyrights holder.
Copies of this book sold without a Wiley sticker on the cover are unauthorized and illegal.
本书中文简体字版由 Wiley Publishing, Inc. 授权清华大学出版社出版。未经出版者书面许可，不得以任何方式复制或传播本书内容。

本书封面贴有 Wiley 公司防伪标签，无标签者不得销售。
版权所有，侵权必究。举报：010-62782989，beiqinquan@tup.tsinghua.edu.cn。

图书在版编目(CIP)数据

深入理解现代 JavaScript / (美)T. J. 克罗德(T. J. Crowder) 著；赵永，卢贤泼译. —北京：清华大学出版社，2022.3
(Web 开发与设计)
书名原文：JavaScript: The New Toys
ISBN 978-7-302-60211-8

I. ①深… II. ①T… ②赵… ③卢… III. ① JAVA 语言—程序设计 IV. ①TP312.8

中国版本图书馆 CIP 数据核字(2022)第 033306 号

责任编辑：王　军
装帧设计：孔祥峰
责任校对：成凤进
责任印制：朱雨萌

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-83470000　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：大厂回族自治县彩虹印刷有限公司
经　　销：全国新华书店
开　　本：170mm×240mm　　印　　张：32　　字　　数：903 千字
版　　次：2022 年 4 月第 1 版　　印　　次：2022 年 4 月第 1 次印刷
定　　价：128.00 元

产品编号：086657-01

译 者 序

回想起来，我对 Web 技术的接触源于 10 年前观看的一场电影——《社交网络》。电影中的“Facemesh”网站由于有趣的创意和交互而在一夜之间风靡校园，令哈佛大学的服务器几近崩溃，这让我深受触动。JavaScript 就在那时闯入了我的世界。

JavaScript 是一门快速变化的语言，它于 1995 年面世，至今已有 26 年的历史，但真正得到广泛的应用是在 2009 年 12 月第 5 版(ES5)发布之后。JavaScript 最初只能用来在网页上编写简单交互的脚本，但现在已经可应用于服务端开发、嵌入式开发、桌面应用程序、机器学习、操作系统等领域。总之，你能想到的场景，JavaScript 都可以实现(阿特伍德定律)。非常幸运，我亲身经历了 JavaScript 蓬勃发展的这 10 年。

本书作者 T. J. Crowder 结合自己多年的 JavaScript 开发经验，将理论和实践很好地结合起来。本书不仅让读者了解过去几年中 JavaScript 增加的新特性，还给出了丰富的示例来呈现这些特性的实际应用，给读者以启发和引导。本书覆盖了 ES2015~ES2020 中所有的新特性，也介绍了 ES.next 中的部分新特性。这是一本非常棒的工具书，值得你经常翻阅。本书基本涵盖了开发人员在日常开发过程中可能涉及的所有 JavaScript 特性。相信你阅读完本书后，会更加熟悉 JavaScript 的功能与特性。其实，本书每个章节的内容都非常丰富并可以单独成书，尤其是第 16 章(共享内存)。在控制篇幅的同时，作者力求在介绍每个特性时做到深入浅出、通俗易懂。

对于译者而言，翻译一本书的过程既是再次学习的过程，也是循着作者的思路提出问题、分析问题、解决问题的思维过程。在本书翻译过程中，我力图把作者的意图真实地呈现给读者，让读者也能感受到作者在本书中所展现的思维。

除本书署名译者外，参与本书翻译的还有王金全等人，感谢他们的辛勤付出。同时要感谢清华大学出版社给予我们这样一个难得的机会，以及在本书翻译过程中给予的指导和帮助。

赵永

2021 年 10 月于北京

作 者 简 介

T. J. Crowder 是一位拥有三十年专业经验的软件工程师，在他的职业生涯中，至少有一半时间是在使用 JavaScript 从事开发工作。他经营着 Farsight Software 公司—— 英国的一家软件咨询和产品公司。作为 Stack Overflow 的十大贡献者之一和 JavaScript 标签的顶级贡献者，他喜欢用所学知识帮助他人解决技术挑战。他不仅授人以鱼，而且授人以渔。

T. J.十几岁时就开始在加利福尼亚编程，使用 BASIC 和汇编语言捣鼓 Apple II、Sinclair ZX-80 和 ZX-81。多年后，他找到了第一份与计算机相关的工作，在一家为法庭书记员提供 PC(DOS)产品的公司担任技术支持。在做技术支持期间，他通过办公室里的 TurboC 手册自学了 C 语言，对公司未记录的压缩二进制文件格式进行了逆向工程，并在业余时间利用这些知识为产品创建了一个备受欢迎的功能(公司很快就发布了该功能)。不久之后，开发部门接纳了他，并在接下来的几年里给了他机会、资源和责任，使他成长为一名程序员，并最终成为开发部门的首席工程师。他创建了各种产品，包括他们的第一个 Windows 产品。

来到下一家公司后，他承担起专业服务和开发人员教育的工作，为客户现场定制公司的企业产品(使用 VB6、JavaScript 和 HTML)，并为客户的开发人员提供培训；最后他还制作了培训课程。从美国搬到伦敦后，他重新担任了直接的开发者角色，在这期间他提高了自己的软件设计能力、SQL、Java 和 JavaScript 技能，然后开始独立承包项目。

从那时起，他主要为各种公司和组织(北约机构、英国地方政府机构和各种私营公司)进行闭源远程开发工作，主要使用 JavaScript、SQL、C#和 TypeScript(最近)。对开源社区的热爱使他先后加入了当年的 PrototypeJS 邮件列表、Stack Overflow，以及现在的各种平台。

T. J.是英裔美国人，在美国出生并长大，现在与妻子和儿子住在英格兰中部的一个村庄。

技术编辑简介

Chaim Krause 是一位专业的计算机程序员，拥有超过三十年的开发经验。早在 1995 年，他就担任过 ISP 的首席技术支持工程师，在 Borland 公司担任 Delphi 的高级开发支持工程师，并在硅谷工作了十多年，担任过各种职务，包括技术支持工程师和开发支持工程师。他目前是美国陆军司令部和总参谋部学院的军事模拟专家，致力于开发用于训练演习的严肃游戏等项目。他还制作了多个 Linux 主题的视频培训课程，并为二十多本书籍担任技术审稿人。

技术校对简介

Marcia K. Wilbur 是一位在半导体领域提供咨询服务的技术传播者，专注于工业物联网(Industrial IoT，IIoT)和 AI。Marcia 拥有计算机科学、技术传播和信息技术学位。作为 Copper Linux 用户组的主席，她积极协助领导 West Side Linux + Pi 的创客社区以及 regular Raspberry Pi、Beaglebone、Banana Pi/Pro 和 ESP8266 的 East Valley 创客社区，涉及的项目包括家庭自动化、游戏机、监控、网络、多媒体以及其他“有趣的事物”。

致　　谢

“我写了一本书”并不是一个准确的说法。

当然，书中所有未引用的文字都是我写的。但如果没有下面提及的这些人，本书根本不会顺利面世：那些给我提供支持、观点和鼓励的人；那些审阅草稿，帮助取其精华、去其糟粕的人；那些多年来在各种论坛上提问题的人(是他们让我有机会展示我所获得的浅薄知识)；那些直接或间接提供资源来回答我问题的人；那些帮助我推敲文字，提高我文字技巧的人。

总之，我应该感谢很多人。这是我撰写的第一本正式的书，我几乎犯了所有可能犯的错误。我要感谢下面这些人，他们已经尽可能多地帮我发现并纠正了书中的错误。

特别感谢妻子 Wendy 对我的支持，感谢她自愿给予我无尽的力量和鼓励，感谢她在我偏离主题时把我引回来(正如 James Taylor 的歌词里所说的)。

感谢我的儿子 James，他忍受了长时间没有父亲陪伴的苦恼。

感谢我的母亲 Virginia 和父亲 Norman，他们引导我爱上了语言、学习、阅读和写作，这些都是持续一生的礼物。

感谢我最好的朋友 Jock，他一直耐心地为我提供意见并给予我很多的鼓励。而且，是他首先让我对编程产生了兴趣。

感谢 Wiley & Sons 的所有编辑和审稿人。虽然作为作者的我犯了很多大大小小的错误，Jim Minatel 和 Pete Gaughan 仍然支持和维护这个项目；David Clark 和 Chaim Krause 对本书进行了编辑和技术审查；Kim Cofer 进行了出色的文字编辑，并在语法、句法和清晰度方面提供了帮助；Nancy Bell 对本书进行了校对。也感谢制作部门的设计师和排版人员，以及我没有提到名字的，在各个方面给这个项目提供支持的那些人。

感谢 Andreas Bergmaier 帮助我保持技术细节的正确性，Andreas 敏锐的眼光和深刻的理解在整本书中都有很大的帮助。

感谢来自 Igalia 的 TC39 成员 Daniel Ehrenberg，他帮助我更好地理解 TC39 的工作方式，并善意地帮助我审查和完善第 1 章。他温和的纠正、投入和洞察力极大地改进了这一章。

感谢来自 Mozilla 的 TC39 成员 Lars T. Hansen(JavaScript 共享内存和 Atomics 提案的共同作者)，他帮助确保了第 16 章的细节和范围的正确性。他渊博的知识和观点让我们受益匪浅。

最后，感谢你，亲爱的读者，感谢你对我的努力给予时间和关注。希望本书对你有帮助。

前　言

如果你是 JavaScript(或 TypeScript)开发人员，并且想了解在过去几年中被添加到 JavaScript 的最新特性，以及如何在语言不断发展的过程中掌握新动态，那么本书适用于你。只要你努力寻找，并对你信任的网站持谨慎态度，就几乎可以在网上找到本书中的所有内容；本书提供了所有的技术细节，同时告诉你如何跟踪不断发生的变化。

本书内容

下面是每一章的内容概览。

第 1 章，ES2015~ES2020 及后续版本的新特性——首先介绍 JavaScript 世界中的各种角色和一些重要的术语；然后描述“新特性”在本书中的定义，以及将新特性添加到 JavaScript 的流程，包括这个流程是如何管理的，由谁管理，以及如何跟踪和参与这一流程；最后介绍一些在旧环境中使用新特性所需的工具(或在当前环境中使用最新特性所需的工具)。

第 2 章，块级作用域声明：let 和 const——涵盖新的声明关键字 let 和 const 以及它们支持的新作用域，深入介绍循环中的作用域，重点说明 for 循环中作用域的处理。

第 3 章，函数的新特性——涵盖与函数有关的各种新特性：箭头函数、默认参数值、“rest”参数、name 属性和其他的语法改进。

第 4 章，类——涵盖新的 class 特性：基本概念、子类、super、创建内置对象(如 Array 和 Error)的子类，以及 new.target 特性。私有字段和其他处于提案流程中的特性将在第 18 章介绍。

第 5 章，对象的新特性——涵盖可计算属性名、属性的简写语法、获取和设置对象的原型、新的 Symbol 类型以及它与对象的关系、方法语法、属性顺序、属性的展开语法，以及大量新的对象方法。

第 6 章，可迭代对象、迭代器、for-of 循环、可迭代对象的展开语法和生成器——涵盖迭代(一种强大的用于集合和列表的新工具)，以及生成器(一种强大的与函数交互的新方式)。

第 7 章，解构——涵盖解构这一重要的新语法，以及如何使用它从对象、数组和其他可迭代对象中提取数据，该章包含默认值、嵌套提取等语法。

第 8 章，Promise——深入研究这个用于处理异步过程的重要新工具。

第 9 章，异步函数、迭代器和生成器——详细介绍新的 async/await 语法(它允许你在异步代码中使用熟悉的逻辑流结构)，以及异步迭代器和生成器的工作方式，还有新的 for-await-of 循环。

第 10 章，模板字面量、标签函数和新的字符串特性——描述模板字面量语法、标签函数和许多新的字符串特性，如更好的 Unicode 支持、常见方法的更新以及很多新方法。

第 11 章，新数组特性、类型化数组——涵盖很多新的数组方法、各种已有方法的更新、类型化数组(如 Int32Array)以及与类型化数组数据交互的高级特性。

第 12 章，Map 和 Set——介绍所有新的有键集合 Map 和 Set，以及这些集合的“弱”版本 WeakMap 和 WeakSet。

第 13 章，模块——深入了解这个令人兴奋且强大的代码组织方式。

第 14 章，反射和代理——涵盖 Reflect 和 Proxy 对象的强大动态元编程特性以及它们之间的关系。

第 15 章，正则表达式更新——描述过去几年正则表达式出现的所有更新，如新的标志、命名捕获组、反向预查和新的 Unicode 特性。

第 16 章，共享内存——涵盖 JavaScript 程序中有关跨线程共享内存的复杂而棘手的方面，其中包括 SharedArrayBuffer 和 Atomics 对象、基本概念和陷阱注释。

第 17 章，其他特性——涵盖很多不适合放到其他章节的新特性：BigInt、新的整数字面量语法(二进制、新的八进制)、省略 catch 绑定的异常、新的 Math 方法、取幂运算符、Math 对象的扩展、尾递归优化、空值合并、可选链，以及出于兼容性原因而定义的“规范附录 B”(仅浏览器)特性。

第 18 章，即将推出的类特性——描述在提案流程中处于阶段 3 的类的增强特性：公有字段声明、私有字段和私有方法。

第 19 章，展望未来——最后，描述目前正在进行的一些改进：顶层 await、WeakRef 和清理回调、正则表达式匹配索引、Atomics.asyncWait、一些新的语法特性、旧的正则表达式特性，以及各种即将推出的标准库扩展。

附录，出色的特性及对应的章(向 J. K. Rowling 致歉)——提供新特性的列表，并指出每个特性所属的章节。这些列表包括：按字母顺序排列的特性，新的基础知识，新的语法、关键字、运算符、循环等，新的字面量形式，标准库的扩展和更新，以及其他特性。

本书读者对象

本书的读者应该：

- 至少对 JavaScript 有基本的了解。
- 想了解过去几年中增加的新特性。

这不是一本为专家编写的学术书籍，而是一本面向 JavaScript 开发人员的实用性书籍。

几乎所有拿起本书的人都知道书中的一些内容，但几乎没有人拿起这本书时就已经知道了所有内容。也许你已经清楚 let 和 const 的基础知识，但是还没有完全掌握 async 函数。也许 Promise 对你来说已经是旧语法了，但你在一些现代代码中看到了一些不认识的语法。你可以在本书中找到 ES2015~ES2020(及后续版本)的所有新特性。

如何使用本书

建议先阅读第 1 章。第 1 章定义了本书其余部分使用的很多术语。如果跳过第 1 章，你很可能会在阅读本书时遇到困难。

之后，你可以选择按顺序阅读各章，或者跳着阅读。

我以这样的顺序安排各章内容是有原因的，而且每个章节都与前面章节息息相关。例如，第 8 章介绍的 Promise，对于理解第 9 章中的 async 函数很重要。当然，建议你按照我安排的顺序阅读本书。不过我敢肯定，你是一个有自己想法的聪明人，如果你不按照顺序阅读，也没关系。

建议你阅读(或者至少略读)所有的章节(第 16 章可能除外，稍后再谈这个问题)。即使你认为自己已了解某个特性，也可能不知道或者只是认为自己知道书中的一些内容。例如，也许你打算跳过第 2 章，因为你已经知道关于 let 和 const 的所有知识。你甚至知道为什么下面的代码创建了 10 个不同的变量 i：

```
for (let i = 0; i < 10; ++i) { /*...*/
   setTimeout(() => console.log(i));
}
```

还有，如果像这样使用：

```
let a = "ay";
var b = "bee";
```

为什么会在全局作用域创建 window.b 属性，却没有创建 window.a 属性？即使这些你都清楚，我也建议你略读第 2 章，以确保你掌握所有内容。

第 16 章有点特殊：它是关于如何在线程之间共享内存的。大多数 JavaScript 开发人员都不需要在线程之间共享内存。但有些开发人员需要，这也是第 16 章存在的原因；而大多数人不需要，如果你属于这一类，则可跳过该章；如果你认为自己在将来某个时候需要共享内存，可再回到第 16 章中学习它，这没关系。

此外，运行本书的示例，用它们进行试验，祝你编程愉快。

本书代码下载

你可以通过网址 https://thenewtoys.dev/bookcode 或 https://www.wiley.com/go/javascript-newtoys 下载各章的示例和代码清单，也可通过扫描封底的二维码下载。

目　录

第 1 章　ES2015~ES2020 及后续版本的新特性 ……1
1.1　名称、定义和术语 ……2
1.1.1　Ecma？ECMAScript？TC39? ……2
1.1.2　ES6？ES7？ES2015？ES2020？ ……2
1.1.3　JavaScript“引擎”、浏览器及其他……3
1.2　什么是“新特性” ……3
1.3　新特性的推动流程 ……5
1.3.1　谁负责 ……5
1.3.2　流程 ……5
1.3.3　参与 ……6
1.3.4　跟上新特性的步伐 ……7
1.4　旧环境中使用新特性 ……8
1.5　本章小结 ……12

第 2 章　块级作用域声明：let 和 const ……13
2.1　let 和 const 的介绍 ……13
2.2　真正的块级作用域 ……14
2.3　重复声明将抛出错误 ……15
2.4　提升和暂时性死区 ……15
2.5　一种新的全局变量 ……17
2.6　const：JavaScript 的常量 ……19
2.6.1　const 基础 ……19
2.6.2　常量引用的对象仍然是可变的 ……20
2.7　循环中的块级作用域 ……21
2.7.1　“循环中的闭包”问题 ……21
2.7.2　绑定：变量、常量以及其他标识符的工作方式 ……23
2.7.3　while 和 do-while 循环 ……27
2.7.4　性能影响 ……28
2.7.5　循环块中的 const ……29
2.7.6　for-in 循环中的 const ……29
2.8　旧习换新 ……30
2.8.1　用 const 或 let 替代 var ……30
2.8.2　缩小变量的作用域 ……30
2.8.3　用块级作用域替代匿名函数 ……30

第 3 章　函数的新特性 ……33
3.1　箭头函数和 this、super 等词法 ……34
3.1.1　箭头函数语法 ……34
3.1.2　箭头函数和 this 词法 ……37
3.1.3　箭头函数不能被用作构造函数 ……38
3.2　默认参数值 ……38
3.2.1　默认值是表达式 ……39
3.2.2　默认值在自己的作用域中被计算 ……40
3.2.3　默认值不会增加函数的 arity ……42
3.3　“rest”参数 ……42
3.4　参数列表和函数调用中的尾后逗号 ……44
3.5　函数的 name 属性 ……45
3.6　在语句块中声明函数 ……46
3.6.1　在语句块中声明函数：标准语义 ……48
3.6.2　在语句块中声明函数：传统 Web 语义 ……49
3.7　旧习换新 ……51
3.7.1　使用箭头函数替代各种访问 this 值的变通方式 ……51
3.7.2　在不使用 this 或 arguments 时，回调函数使用箭头函数 ……52
3.7.3　考虑在更多地方使用箭头函数 ……52
3.7.4　当调用者需要控制 this 的值时，不要使用箭头函数 ……53
3.7.5　使用参数默认值，而不是代码实现 ……53
3.7.6　使用“rest”参数替代 arguments 关键字 ……53
3.7.7　如有必要，考虑使用尾后逗号 ……53

第 4 章　类 ……55
4.1　类的概念 ……55

4.2 介绍新的类语法 …… 56
4.2.1 添加构造函数 …… 57
4.2.2 添加实例属性 …… 59
4.2.3 添加原型方法 …… 59
4.2.4 添加静态方法 …… 61
4.3 添加访问器属性 …… 61
4.4 对比新语法和旧语法 …… 64
4.5 创建子类 …… 66
4.6 关键字 super …… 69
4.6.1 编写子类构造函数 …… 69
4.6.2 继承和访问超类原型的属性和方法 …… 70
4.6.3 继承静态方法 …… 73
4.6.4 静态方法中的 super …… 75
4.6.5 返回新实例的方法 …… 75
4.6.6 内置对象的子类 …… 79
4.6.7 super 的使用 …… 81
4.7 移除 Object.prototype …… 83
4.8 new.target …… 84
4.9 类声明与类表达式 …… 87
4.9.1 类声明 …… 87
4.9.2 类表达式 …… 88
4.10 更多内容 …… 89
4.11 旧习换新 …… 89

第 5 章 对象的新特性 …… 91
5.1 可计算属性名 …… 91
5.2 属性的简写语法 …… 92
5.3 获取和设置对象原型 …… 93
5.3.1 Object.setPrototypeOf …… 93
5.3.2 浏览器环境中的__proto__属性 …… 94
5.3.3 浏览器环境中的__proto__字面量属性名 …… 94
5.4 对象方法的简写语法，以及类之外的 super …… 95
5.5 Symbol …… 97
5.5.1 定义 Symbol 的原因 …… 97
5.5.2 创建和使用 Symbol …… 99
5.5.3 Symbol 并不用于私有属性 …… 99
5.5.4 全局 Symbol …… 100
5.5.5 内置的 Symbol 值 …… 103
5.6 对象的新增方法 …… 104
5.6.1 Object.assign …… 104
5.6.2 Object.is …… 105
5.6.3 Object.values …… 105
5.6.4 Object.entries …… 106
5.6.5 Object.fromEntries …… 106
5.6.6 Object.getOwnPropertySymbols …… 106
5.6.7 Object.getOwnPropertyDescriptors …… 106
5.7 Symbol.toPrimitive …… 107
5.8 属性顺序 …… 109
5.9 属性扩展语法 …… 110
5.10 旧习换新 …… 111
5.10.1 创建对象时对动态变量使用可计算属性名 …… 111
5.10.2 从同名变量初始化对象时，使用简写语法 …… 111
5.10.3 使用 Object.assign 替代自定义的扩展方法或者显式复制所有属性 …… 112
5.10.4 基于已有对象创建新对象时，使用属性扩展语法 …… 112
5.10.5 使用 Symbol 避免属性名冲突 …… 112
5.10.6 使用 Object.getPrototypeOf/setPrototypeOf 替代__proto__ …… 112
5.10.7 使用对象方法的简写语法来定义对象中的方法 …… 112

第 6 章 可迭代对象、迭代器、for-of 循环、可迭代对象的展开语法和生成器 …… 115
6.1 迭代器、可迭代对象、for-of 循环，以及可迭代对象的展开语法 …… 115
6.1.1 迭代器和可迭代对象 …… 115
6.1.2 for-of 循环：隐式地使用迭代器 …… 116
6.1.3 显式地使用迭代器 …… 117
6.1.4 提前停止迭代 …… 118
6.1.5 迭代器的原型对象 …… 119
6.1.6 使对象可迭代 …… 121
6.1.7 使迭代器可迭代 …… 124
6.1.8 可迭代对象的展开语法 …… 126
6.1.9 迭代器、for-of 循环和 DOM …… 127
6.2 生成器函数 …… 129
6.2.1 仅生成值的基本生成器函数 …… 129

6.2.2 使用生成器函数创建迭代器 130
6.2.3 生成器函数作为方法 131
6.2.4 直接使用生成器 132
6.2.5 用生成器消费值 132
6.2.6 在生成器函数中使用 return 136
6.2.7 yield 运算符的优先级 136
6.2.8 return 和 throw 方法：终止生成器 137
6.2.9 生成生成器或者可迭代对象：yield* 139
6.3 旧习换新 143
6.3.1 使用消费可迭代对象的结构 143
6.3.2 使用 DOM 集合的可迭代特性 144
6.3.3 使用可迭代对象和迭代器接口 144
6.3.4 在过去用 Function.prototype.apply 的大部分场景中使用可迭代对象的展开语法 144
6.3.5 使用生成器 144

第 7 章 解构 145
7.1 概览 145
7.2 基础的对象解构 145
7.3 基础的数组(和可迭代对象)的解构 148
7.4 解构默认值 150
7.5 解构匹配模式中的“rest”语法 151
7.6 使用不同的名称 152
7.7 可计算属性名 153
7.8 嵌套解构 153
7.9 参数解构 154
7.10 循环中的解构 157
7.11 旧习换新 157
7.11.1 仅从对象获取某些属性时使用解构 158
7.11.2 对可选项对象使用解构 158

第 8 章 Promise 159
8.1 为什么要使用 Promise 159
8.2 Promise 基础 160
8.2.1 概览 160
8.2.2 示例 161
8.2.3 Promise 和 thenable 对象 163
8.3 使用已存在的 Promise 163
8.3.1 then 方法 163
8.3.2 链式 Promise 164
8.3.3 对比 Promise 链与回调函数 168
8.3.4 catch 方法 168
8.3.5 finally 方法 170
8.3.6 在 then、catch 和 finally 处理程序中抛出异常 173
8.3.7 带有两个参数的 then 方法 175
8.4 为已敲定状态的 Promise 添加处理程序 176
8.5 创建 Promise 177
8.5.1 Promise 构造函数 178
8.5.2 Promise.resolve 180
8.5.3 Promise.reject 181
8.6 其他 Promise 工具方法 182
8.6.1 Promise.all 182
8.6.2 Promise.race 183
8.6.3 Promise.allSettled 184
8.6.4 Promise.any 184
8.7 Promise 的模式 185
8.7.1 处理错误或返回 Promise 185
8.7.2 串行 Promise 185
8.7.3 并行 Promise 187
8.8 Promise 的反模式 188
8.8.1 不必要的 new Promise(/*…*/) 188
8.8.2 未处理的错误(或不正确的处理方式) 188
8.8.3 在转换回调函数 API 时隐藏了错误 188
8.8.4 隐式地将已拒绝状态转换为已成功状态 189
8.8.5 试图在链式调用外使用结果 190
8.8.6 使用无用的处理程序 190
8.8.7 错误地处理链式调用分支 191
8.9 Promise 的子类 192
8.10 旧习换新 193

第 9 章 异步函数、迭代器和生成器 195
9.1 async 函数 195
9.1.1 async 函数创建 Promise 对象 197
9.1.2 await 接收 Promise 198

9.1.3 异步是使用 await 的常规思维方式……199
9.1.4 拒绝即异常，异常即拒绝；成功值就是结果，返回值就是决议……200
9.1.5 async 函数中的并行操作……202
9.1.6 不必使用 return await……203
9.1.7 陷阱：在意想不到的地方使用 async 函数……204
9.2 异步迭代器、可迭代对象和生成器……205
9.2.1 异步迭代器……205
9.2.2 异步生成器……208
9.2.3 for-await-of……209
9.3 旧习换新……210

第 10 章 模板字面量、标签函数和新的字符串特性……211
10.1 模板字面量……211
10.1.1 基本功能(不带标签的模板字面量)……212
10.1.2 模板标签函数(带标签的模板字面量)……213
10.1.3 String.raw……218
10.1.4 模板字面量的复用……219
10.1.5 模板字面量和自动分号插入……219
10.2 改进的 Unicode 支持……219
10.2.1 Unicode 以及 JavaScript 字符串的含义……219
10.2.2 码点转义序列……221
10.2.3 String.fromCodePoint……221
10.2.4 String.prototype.codePointAt……221
10.2.5 String.prototype.normalize……222
10.3 迭代……223
10.4 新的字符串方法……224
10.4.1 String.prototype.repeat……224
10.4.2 String.prototype.startsWith 和 endsWith……224
10.4.3 String.prototype.includes……225
10.4.4 String.prototype.padStart 和 padEnd……225
10.4.5 String.prototype.trimStart 和 trimEnd……226
10.5 match、split、search 和 replace 方法的更新……226
10.6 旧习换新……228
10.6.1 使用模板字面量替代字符串连接(在适当的情况下)……228
10.6.2 对 DSL 使用标签函数和模板字面量，而不是自动占位符机制……228
10.6.3 使用字符串迭代……228

第 11 章 新数组特性、类型化数组……229
11.1 新的数组方法……229
11.1.1 Array.of……229
11.1.2 Array.from……230
11.1.3 Array.prototype.keys……232
11.1.4 Array.prototype.values……233
11.1.5 Array.prototype.entries……233
11.1.6 Array.prototype.copyWithin……234
11.1.7 Array.prototype.find……236
11.1.8 Array.prototype.findIndex……237
11.1.9 Array.prototype.fill……238
11.1.10 Array.prototype.includes……239
11.1.11 Array.prototype.flat……239
11.1.12 Array.prototype.flatMap……240
11.2 迭代、展开、解构……241
11.3 稳定的数组排序……241
11.4 类型化数组……241
11.4.1 概述……242
11.4.2 基本用法……243
11.4.3 ArrayBuffer：类型化数组使用的存储方式……246
11.4.4 Endianness(字节序)……247
11.4.5 DataView：直接访问缓冲区……248
11.4.6 在数组间共享 ArrayBuffer……250
11.4.7 类型化数组的子类……251
11.4.8 类型化数组方法……251
11.5 旧习换新……253
11.5.1 使用 find 和 findIndex 方法替代循环来搜索数组(在适当的情况下)……253
11.5.2 使用 Array.fill 替代循环填充数组……254

11.5.3 使用 readAsArrayBuffer 替代 readAsBinaryString …… 254

第 12 章 Map 和 Set …… 255
12.1 Map …… 255
12.1.1 Map 的基本操作 …… 256
12.1.2 键的相等性 …… 257
12.1.3 从可迭代对象中创建 Map …… 258
12.1.4 迭代 Map 的内容 …… 259
12.1.5 创建 Map 的子类 …… 261
12.1.6 性能 …… 261
12.2 Set …… 262
12.2.1 Set 的基本操作 …… 262
12.2.2 从可迭代对象中创建 Set …… 263
12.2.3 迭代 Set 的内容 …… 263
12.2.4 创建 Set 的子类 …… 265
12.2.5 性能 …… 265
12.3 WeakMap …… 265
12.3.1 WeakMap 是不可迭代的 …… 266
12.3.2 用例与示例 …… 266
12.3.3 值反向引用键 …… 269
12.4 WeakSet …… 274
12.5 旧习换新 …… 276
12.5.1 在通用的映射中使用 Map 替代对象 …… 276
12.5.2 以 Set 替代对象作为集合 …… 277
12.5.3 使用 WeakMap 存储私有数据，而不是公共属性 …… 277

第 13 章 模块 …… 279
13.1 模块简介 …… 279
13.2 模块的基本概念 …… 280
13.2.1 模块说明符 …… 281
13.2.2 基本命名导出 …… 282
13.2.3 默认导出 …… 283
13.2.4 在浏览器中使用模块 …… 284
13.2.5 在 Node.js 中使用模块 …… 287
13.3 重命名导出 …… 289
13.4 重新导出另一个模块的导出 …… 290
13.5 重命名导入 …… 291
13.6 导入模块的命名空间对象 …… 292
13.7 导出另一个模块的命名空间对象 …… 292
13.8 仅为副作用导入模块 …… 293
13.9 导入和导出条目列表 …… 293
13.9.1 导入条目列表 …… 293
13.9.2 导出条目列表 …… 294
13.10 导入是实时且只读的 …… 295
13.11 模块实例具有领域特性 …… 297
13.12 模块的加载方式 …… 298
13.12.1 获取和解析 …… 299
13.12.2 实例化 …… 302
13.12.3 执行 …… 302
13.12.4 暂时性死区(TDZ)回顾 …… 303
13.12.5 循环依赖和 TDZ …… 303
13.13 导入/导出语法回顾 …… 304
13.13.1 不同的导出语法 …… 304
13.13.2 不同的导入语法 …… 305
13.14 动态导入 …… 306
13.14.1 动态导入模块 …… 306
13.14.2 动态模块示例 …… 308
13.14.3 非模块脚本中的动态导入 …… 311
13.15 摇树 …… 312
13.16 打包 …… 314
13.17 导入元数据 …… 314
13.18 worker 模块 …… 315
13.18.1 将 Web worker 加载为模块 …… 315
13.18.2 将 Node.js worker 加载为模块 …… 316
13.18.3 每个 worker 都在自己的领域中 …… 316
13.19 旧习换新 …… 316
13.19.1 使用模块替代伪命名空间 …… 317
13.19.2 使用模块替代作用域函数 …… 317
13.19.3 使用模块避免巨石代码文件的创建 …… 317
13.19.4 将 CJS、AMD 和其他模块格式转换为 ESM …… 318
13.19.5 使用维护良好的打包器，而不是自研 …… 318

第 14 章 反射和代理 …… 319
14.1 反射 …… 319
14.1.1 Reflect.apply …… 320
14.1.2 Reflect.construct …… 321

14.1.3 Reflect.ownKeys……322
14.1.4 Reflect.get 和 Reflect.set……322
14.1.5 其他 Reflect 函数……324
14.2 代理……324
14.2.1 示例：日志代理……326
14.2.2 代理劫持函数……334
14.2.3 示例：隐藏属性……342
14.2.4 可撤销代理……345
14.3 旧习换新……346
14.3.1 使用代理，而不是禁止消费侧代码修改 API 对象……346
14.3.2 使用代理将实现代码与检测代码分开……346

第 15 章 正则表达式更新……347
15.1 flags 属性……347
15.2 新标志……348
15.2.1 粘连标志(y)……348
15.2.2 Unicode 标志(u)……349
15.2.3 dot all 标志(s)……349
15.3 命名捕获组……349
15.3.1 基本功能……350
15.3.2 反向引用……353
15.3.3 替换符号……354
15.4 反向预查……354
15.4.1 反向肯定预查……354
15.4.2 反向否定预查……355
15.4.3 反向预查中的贪婪匹配是从右到左的……356
15.4.4 捕获组的编号和引用……356
15.5 Unicode 特性……357
15.5.1 码点转义……357
15.5.2 Unicode 属性转义……358
15.6 旧习换新……361
15.6.1 在解析时使用粘连标志(y)，而不是创建子字符串并使用插入符(^)……361
15.6.2 使用 dot all 标志(s)，而不是使用一些变通方法匹配所有的字符(包括换行符)……361
15.6.3 使用命名捕获组替代匿名捕获组……362
15.6.4 使用反向预查替代各种变通方法……362
15.6.5 在正则表达式中使用码点转义替代代理对……363
15.6.6 使用 Unicode 模式替代变通方法……363

第 16 章 共享内存……365
16.1 引言……365
16.2 务必谨慎……365
16.3 浏览器的支持……366
16.4 共享内存的基础知识……368
16.4.1 临界区、锁和条件变量……368
16.4.2 创建共享内存……369
16.5 共享的是内存，而不是对象……373
16.6 竞争条件、存储乱序、旧值、撕裂等……374
16.7 Atomics 对象……375
16.7.1 Atomics 对象的底层特性……378
16.7.2 使用 Atomics 对象挂起和恢复线程……379
16.8 共享内存示例……380
16.9 务必谨慎(再次)……399
16.10 旧习换新……404
16.10.1 使用共享内存块，而不是重复交换大数据块……404
16.10.2 使用 Atomics.wait 和 Atomics.notify，而不是拆解 worker 任务以支持事件循环(在适当的地方)……404

第 17 章 其他特性……405
17.1 BigInt……406
17.1.1 创建 BigInt……406
17.1.2 显式转换和隐式转换……407
17.1.3 性能……408
17.1.4 BigInt64Array 和 BigUint64Array……408
17.1.5 工具函数……409
17.2 新的整数字面量……409
17.2.1 二进制整数字面量……409
17.2.2 八进制整数字面量，采用 ES2015 新形式……410

17.3 新的 Math 方法……410
17.3.1 通用数学函数……410
17.3.2 提供底层操作的数学函数……411
17.4 取幂运算符(**)……412
17.5 Date.prototype.toString 调整……413
17.6 Function.prototype.toString 调整……413
17.7 Number 扩展……414
17.7.1 “安全”整数……414
17.7.2 Number.isInteger……415
17.7.3 Number.isFinite 和 Number.isNaN……415
17.7.4 Number.parseInt 和 Number.parseFloat……416
17.7.5 Number.EPSILON……416
17.8 Symbol.isConcatSpreadable……416
17.9 其他语法微调……417
17.9.1 空值合并……417
17.9.2 可选链……418
17.9.3 省略 catch 绑定的异常……420
17.9.4 JSON 中的 Unicode 行终止符……420
17.9.5 JSON.stringify 输出符合语法规则的 JSON……420
17.10 标准库/全局对象的各类扩展……421
17.10.1 Symbol.hasInstance……421
17.10.2 Symbol.unscopables……421
17.10.3 globalThis……422
17.10.4 Symbol 的 description 属性……422
17.10.5 String.prototype.matchAll……423
17.11 规范附录 B：浏览器相关特性……423
17.11.1 类似 HTML 的注释……424
17.11.2 正则表达式微调……424
17.11.3 额外的内置属性……425
17.11.4 各种松散或晦涩的语法片段……427
17.11.5 当 document.all 不存在…… 或存在……428
17.12 尾调用优化……429
17.13 旧习换新……431
17.13.1 使用二进制字面量……431
17.13.2 使用新的 Math 函数，而不是各类数学变通方法……432
17.13.3 使用空值合并提供默认值……432
17.13.4 使用可选链替代&&检查……432
17.13.5 省略“catch(e)”中的异常绑定……432
17.13.6 使用取幂运算符(**)，而不是 Math.pow……433

第 18 章 即将推出的类特性……435
18.1 公有和私有的类字段、方法和访问器……435
18.1.1 公有字段(属性)定义……436
18.1.2 私有字段……440
18.1.3 私有实例方法和访问器……446
18.1.4 公有静态字段、私有静态字段和私有静态方法……450
18.2 旧习换新……452
18.2.1 使用属性定义，而不是在构造函数中创建属性(在适当的情况下)……452
18.2.2 使用私有类字段，而不是前缀(在适当的情况下)……453
18.2.3 使用私有方法(而不是类外的函数)进行私有操作……453

第 19 章 展望未来……457
19.1 顶层 await……458
19.1.1 概述和用例……458
19.1.2 示例……459
19.1.3 错误处理……463
19.2 WeakRef 和清理回调……464
19.2.1 WeakRef……464
19.2.2 清理回调……466
19.3 正则表达式匹配索引……471
19.4 String.prototype.replaceAll……472
19.5 Atomics 的 asyncWait 方法……472
19.6 其他语法微调……473
19.6.1 数字分隔符……473
19.6.2 支持 hashbang……474
19.7 废弃旧的正则表达式特性……474
19.8 感谢阅读……475

附录 出色的特性及对应的章(向 J.K. Rowling 致歉)……477

第1章

ES2015~ES2020 及后续版本的新特性

本章内容

- 名称、定义和术语
- JavaScript 的版本说明(ES6？ES2020？)
- “新特性”说明
- JavaScript 新特性的推动流程
- 用于下一代 JavaScript 的工具

本章代码下载

可通过网址 https://thenewtoys.dev/bookcode 或 http://www.wiley.com/go/javascript- newtoys 下载本章的代码。

在过去的几年中，JavaScript 发生了很大的变化。

如果你在 21 世纪初是一个活跃的 JavaScript 开发者，那么也许会认为 JavaScript 是一门停滞不前的语言，这情有可原。在第 3 版规范于 1999 年 12 月发布之后，开发者等了整整 10 年才等到下一版规范。从表面看，似乎确实什么也没有发生。事实上，很多工作都在进行，只是没有被纳入官方规范和多个 JavaScript 引擎中。我们可以花一整章(甚至整本书)的篇幅介绍在 JavaScript 发展过程中扮演重要角色的不同组织做了哪些工作，以及为什么在很长一段时间内，他们对 JavaScript 的发展方向无法达成共识，但我们不会那么做。关键是，经过大量谈判之后，他们最终于 2008 年 7 月在奥斯陆举行的一次重要会议上达成了共识。Brendan Eich(JavaScript 的创建者)后来将那次共识称为 Harmony，这为 2009 年 12 月的第 5 版规范(第 4 版从未完成)铺平了道路，并为持续的进步奠定了基础。

而我的这本书，就是从这个版本开始的。

本章将概述自 2009 年以来 JavaScript 的新特性(本书其余部分将详细介绍这些新特性)。你将了解谁负责推动 JavaScript 的发展、推动的流程，以及下一代的新特性。如果你愿意，可参与进来。本章也会介绍可用来编写现代 JavaScript 的工具，但你必须适配还不支持新特性的环境。

1.1 名称、定义和术语

在谈论 JavaScript 前，本节将首先定义一些名称和常用术语。

1.1.1 Ecma? ECMAScript? TC39?

所谓的 JavaScript 是指由负责多个计算标准的标准组织 Ecma International[1]制定的 ECMAScript 标准化。ECMAScript 的标准是 ECMA-262。负责该标准的人是 TC39 技术委员会(Ecma International 的 Technical Committee 39)的成员，负责“通用、跨平台、与供应商无关的编程语言 ECMAScript 的标准化，这包括语言的语法、语义，以及支持该语言的库和补充技术”[2]。他们也管理其他标准，如 JSON 语法规范(ECMA-404)，尤其是 ECMAScript 国际化 API 规范(ECMA-402)。

在本书和习惯用语中，JavaScript 就是 ECMAScript，反之亦然。有时，特别是在没有统一规范的 10 年间，JavaScript 被用来特指 Mozilla 开发的一门语言(其中有一些功能从未纳入 ECMAScript 规范，或在此之前进行了显著的更改)，但是自从 Harmony 出现之后，这种用法已经日渐过时了。

1.1.2 ES6? ES7? ES2015? ES2020?

这么多的缩写可能会让人困惑，因为其中有些是版本号，而有些是年份，人们对此愈发迷惑不解了。本节将说明这些缩写的含义以及为什么有两种类型。

直到第 5 版，TC39 才通过版本号提及规范的各个版本。第 5 版规范的全称是：

```
Standard ECMA-262
5th Edition / December 2009
ECMAScript Language Specification
```

因为“ECMAScript 5th Edition”有点拗口，所以它自然而然地被说成“ES5”。从 2015 年第 6 版开始，TC39 采用了持续改进的流程，其中，规范是一个长期维护的编辑草案[3]，每年都有快照(随后会详细介绍)。同时，他们在语言名称中加入了年份，如下所示：

```
Standard ECMA-262
6th Edition / June 2015
ECMAScript® 2015 Language Specification
```

因此，ECMAScript 第 6 版标准(“ES6”)定义了 ECMAScript 2015，简称“ES2015”。在 ES2015 发布之前，人们普遍采用 ES6 的说法且一直持续至今(遗憾的是，它经常被不准确地使用，不仅指 ES2015 的特性，还指 ES2016、ES2017 等之后版本的特性)。

这就解释了为什么有两种类型的缩写：一种使用版本号(ES6、ES7 等)的方式，一种使用年份的方式(ES2015、ES2016 等)。具体使用哪种方式，取决于你。ES6 指的是 ES2015(或有时被错误地称为 ES2015 +)，ES7 指的是 ES2016，ES8 指的是 ES2017，以此类推，直到 ES11(在本书发布时)，它指的是 ES2020。你还会看到“ESnext”或“ES.next”之类的缩写，它们有时用于指即将到来的特性。

在本书中，ES5 及以前的版本采用版本号的表示方式，ES2015 及后续版本采用年份的表示方式。这在业界已经日益达成了共识。

---

1 前身为欧洲计算机制造商协会(European Computer Manufacturer’s Association，ECMA)，但现在该组织的名称中只有 Ecma 中的 E 大写。

2 http://www.ecma-international.org/memento/TC39.htm。

3 https://tc39.es/ecma262/。

总之，虽然本书在介绍某个特性的时候通常会说明具体版本，但请不要太在意一些事实，比如：Array.prototype.includes 来自 ES2016，而 Object.values 来自 ES2017。更重要的是：你的目标环境支持什么特性，以及在不支持某个特性时是选择避免使用，还是通过编译或 polyfill 使用。(稍后在 1.4 节“旧环境中使用新特性”会有更多关于编译和 polyfill 的内容。)

1.1.3　JavaScript“引擎”、浏览器及其他

本书将使用“JavaScript 引擎”一词代指运行 JavaScript 代码的软件。JavaScript 引擎必须具备以下几种能力：

- 解析 JavaScript；
- 可解释 JavaScript 或将其编译为机器码(或都支持)；
- 运行结果符合规范中的描述。

JavaScript 引擎有时也被称为虚拟机，或简称为 VM。

当然，JavaScript 引擎通常位于 Web 浏览器中：

- Google 的 Chrome 浏览器使用其 V8 引擎(Chromium、Opera 和微软 Edge 的 v79 及后续版本也使用该引擎)，但不在 iOS 上使用(稍后会详细介绍)。
- 苹果公司的 Safari 浏览器(用于 Mac OS 和 iOS)使用其 JavaScriptCore 引擎。
- Mozilla 的 Firefox 使用其 SpiderMonkey 引擎，但不在 iOS 上使用。
- 微软的 IE 浏览器使用其 JScript 引擎，该引擎已逐渐过时，仅能获得安全修复。
- 微软 Edge v44 及更早的版本(“旧 Edge”)使用微软的 Chakra 引擎。2020 年 1 月，Edge v79 发布，该版本基于 Chromium 项目并使用了 V8 引擎，但不在 iOS 上使用(版本号从 44 跃升到 79，以便与 Chromium 保持一致)。Chakra 仍被用于使用微软 WebView 控件的各种产品，例如微软 Office 的 JavaScript 插件，尽管它可能在某个阶段被取代。(以 Chromium Edge 作为引擎的 WebView2 在 2020 年初处于开发者预览版。)

在苹果公司的 iPad 和 iPhone 的 iOS 操作系统上运行的 Chrome、Firefox、Edge 和其他浏览器目前无法使用自己的 JavaScript 引擎，因为要编译并运行 JavaScript(而不仅仅是对其进行解释)，它们必须分配可执行内存，而只有苹果公司自己的 iOS 应用程序可以这样做，其他供应商的应用程序则无权执行此操作。因此，尽管 Chrome 和 Firefox(及其他)在台式机和 Android 上使用自己的引擎，但在 iOS 上使用苹果的 JavaScriptCore。至少目前是这样；V8 团队在 2019 年为 V8 添加了“仅解释器”模式，这意味着 Chrome 和其他使用 V8 的公司可在 iOS 上使用该模式，因为它不必使用可执行内存。本书中的“Chrome 支持”或“Firefox 支持”分别指的是使用 V8 或 SpiderMonkey 的非 iOS 版本。

JavaScript 引擎还被用于桌面应用程序(Electron[1]、React Native[2] 和其他)、Web 服务器和其他类型的服务器(通常使用 Node.js[3])、非 Web 应用程序、嵌入式应用程序——几乎无处不在。

1.2　什么是“新特性”

在本书中，“新特性”是指在 ES2015~ES2020 中添加到 JavaScript 的特性(包含一部分即将到来的特性)。在这 6 次更新中，JavaScript 有了很大的进步。下面给出了一个总体概述(规范附录 A 有一个更完整的更新列表)。你可能不熟悉列表中的部分术语，不必担心，你将在本书的学习过程中了解它们。

1 https://www.electronjs.org/。

2 https://reactnative.dev/。

3 https://nodejs.org/。

- 块作用域(let，const)：变量作用域更小，对 for 循环中作用域的巧妙处理，值不能改变的“变量”(const)。
- “箭头”函数：轻量、简洁的函数，在回调函数场景特别有用，因为箭头函数对 this 进行了封装，而不是在调用时设置 this 值。
- 函数参数的改进：默认值、参数解构、“rest”参数、尾后逗号。
- 可迭代对象：为创建和消费可迭代对象(如数组和字符串)定义了清晰的语义，语言中的迭代结构(for-of、for-await-of)，用于生成可迭代序列(包括异步序列)的生成器函数。
- “展开”语法：将数组(或其他可迭代对象)条目展开到新的数组中，将对象属性展开到新的对象中，将可迭代对象的条目展开成函数离散参数，特别适合用于函数式编程或其他使用不可改变的结构的地方。
- “rest”语法：把一个对象的“剩余”属性、一个可迭代对象的“剩余”值或一个函数的“剩余”参数合成一个对象或数组。
- 其他语法改进：调用函数时允许在参数列表中使用尾后逗号；在 catch 中省略未使用的标识符；新的八进制字面量；二进制字面量；数字字面量中的分隔符；等等。
- 解构：用比对象和数组字面量语法更简洁的方式从数组/对象中选值。
- 类：创建构造函数和相关原型对象的明显更简单的声明式语法，同时保留了 JavaScript 固有的原型本质。
- 异步编程的改进：Promise、async 函数和 await；显著地减少了“回调地狱”。
- 对象字面量的改进：可计算属性名、属性的简写语法、方法语法、属性定义后的尾后逗号。
- 模板字面量：创建有动态内容的字符串的一种简单、声明式的语法，也可用于创建更强大的标签模板函数。
- 类型化数组：为使用本地 API(以及更多能力)而建立的底层真正的数组。
- 共享内存：在 JavaScript 线程(包括线程间协调原语)之间真正共享内存的能力。
- Unicode 字符串的改进：Unicode 码点转义序列，支持获取码点，而不是代码单元。
- 正则表达式的改进：反向预查，命名捕获组，匹配索引，Unicode 属性转义，Unicode 不区分大小写。
- Map：键/值对集合，其中键不一定是字符串。
- Set：语义清晰的唯一值的集合。
- WeakMap、WeakSet 和 WeakRef：仅持有对象的弱引用的内置对象(允许对象被当作垃圾收集)。
- 标准库扩展：Object、Array、Array.prototype、String、String.prototype、Math 等的新方法。
- 动态元编程的支持：Proxy 和 Reflect。
- Symbol：保证唯一的值(对唯一的属性名特别有用)。
- BigInt：任意精度的整数。
- 还有许多其他的特性。

所有这些新特性，尤其是新语法，可能会让人不知所措和焦虑。不用担心！除非你已经准备好并有实际需求，否则没必要采用新特性。TC39 坚持的主要原则之一是“别毁坏 Web”(Don’t break the web)。这意味着 JavaScript 必须保持“Web 兼容”，也就是与现在已经存在的大量代码兼容[1]。如果你不需要或不喜欢一个新特性，则不必使用它。你使用对应旧语法编写的代码将始终有效。但在很多情况下，你可能会发现你完全有理由使用新特性，尤其是新的语法功能：它们使一些东西的编写和理解变得更简单，更不容易出错，或者在 Proxy 和 WeakMap/WeakSet、共享内存和其他情况下，它们使一些之前

1 委员会也关注与 Web 没有直接关系的重要 JavaScript 代码。

难以实现的功能成为可能。

由于篇幅原因，本书仅涵盖 JavaScript 规范 ECMA-262 中的新特性。但是 ECMA-402 (ECMAScript 国际化 API 规范)中也有一些令人兴奋的新特性，非常值得一读。你可在本书的网站上找到有关 ECMA-402 的部分，网址是 https://thenewtoys.dev/internationalization。

1.3　新特性的推动流程

本节将介绍由谁负责推动 JavaScript 的发展，他们用什么流程来推动，以及如何跟踪和参与这个流程。

1.3.1　谁负责

前面介绍过，TC39 技术委员会负责创建和发布 ECMAScript 标准的更新规范。该委员会的成员包括：JavaScript 开发者、框架作者、大型网站作者/维护人员、编程语言研究人员、所有主要 JavaScript 引擎的代表、有影响力的 JavaScript 编程人员以及其他与 JavaScript 的成功与未来有关的人。他们有定期的会议，历史上每年召开 6 次会议，每次持续 3 天。如果想以成员身份参加会议，你所在的组织需要加入 Ecma[1]。开发者需求、实现复杂度、安全问题、向后兼容性以及许多其他的设计输入都是复杂问题，TC39 会综合考虑这些问题，为 JavaScript 社区设计出有用的新特性。

为确保委员会是社区的一部分而不脱离社区，TC39 在 GitHub 上创建了 ecma262 仓库[2]，其中有最新的规范(可在 https://tc39.es/ecma262/上查看)；它还创建了一个提案仓库[3]，用于维护需要通过 TC39 流程(下一节介绍)的提案。一些成员还活跃在 TC39 论坛小组[4]中。TC39 会议的记录和相关材料(幻灯片等)都会发布在 https://github.com/tc39/notes 上。

你可访问 https://tc39.es/，了解更多关于 TC39 以及如何参与的信息；还可在 https://github.com/tc39/how-we-work 上了解更多关于 TC39 工作方式的信息。

1.3.2　流程

TC39 在 2013 年 11 月通过了一个定义明确的流程，并在 2014 年 1 月首次公布。该流程有多个阶段，TC39 使提案经历这些阶段(从阶段 0 到阶段 4)。每个阶段都有明确的预期，并明确指出从一个阶段进入下一个阶段的标准。一旦一个提案符合进入下一阶段的标准，委员会就会以协商一致的方式决定是否将其向前推进。

流程文档[5]本身值得一读，但简单而言，共有以下这些阶段。

- 阶段 0——Strawperson：任何觉得自己的想法值得考虑的人都可将它写下来，然后提出来。这几乎不能被称为一个阶段，因为想法可能随时变化。如果提出建议的人不是 TC39 成员，则需要注册为非成员贡献者[6](任何人都可这样做)。最终，只有一部分提案会被列入 TC39 的提案仓库中，而有些则不会，通常只有那些获得委员会成员支持的提案才能进入提案仓库。

1 https://ecma-international.org/memento/join.htm。

2 https://github.com/tc39/ecma262。

3 https://github.com/tc39/proposals。

4 https://es.discourse.group/。

5 https://tc39.es/process-document/。

6 https://tc39.es/agreements/contributor/。

如果这一阶段的提案引起了足够的关注，TC39 的成员可能会把它加入 TC39 会议的议程中，以讨论并考虑是否使其进入阶段 1。

- 阶段 1——Proposal：如果一个提案被提交给委员会，且委员会一致认为应进一步调查，他们就会将其转入阶段 1，并指定一名评委来指导整个流程。如果该提案尚无 GitHub 仓库，则需要由发起人、评委或其他感兴趣的第三方创建一个。然后，社区成员(无论是否在委员会中)将进一步讨论和补充该想法，研究其他语言或环境中类似的技术，细化范围，找出通用解决方案，并充实这个想法。这一步的结果可能是，该提案收益比偏低，不值得继续推进；或者该想法需要分解并添加到其他提案中；等等。但是，如果该提案有价值，相关人员(可能随着时间的推移人员会发生变化)将把一些初步的规范语言、API 和语义草案一并提交给 TC39，以便他们考虑是否进入阶段 2。
- 阶段 2——Draft：准备就绪后，相关人员将在 TC39 会议上提交阶段 1 的提案，审议是否进入阶段 2。这意味着 TC39 要就该提案是否继续进行达成共识，并期望该提案最终被纳入规范。在这个阶段，社区会完善精确的语法、语义、API 等并使用正式的规范语言详细描述解决方案。通常，相关人员会在这个阶段创建 polyfill 或 Babel 插件，以便实际试用。提案可能会在阶段 2 停留一段时间，因为需要解决一些细节问题。
- 阶段 3—— Candidate：一旦团队确定了提案的最终草案形式并为其创建了正式的规范语言，评委就可将其提交到阶段 3。这意味着 TC39 会就该提案是否已准备好在 JavaScript 引擎中实现这一问题寻求共识。在这个阶段，提案本身几乎是稳定的。此时的更改将仅限于对该提案实现的反馈，例如在实现过程中发现的极端情况、Web 兼容性问题或实现复杂度。
- 阶段 4—— Finished：至此，该特性已完成，可被添加到编辑草案中，网址是 https://tc39.es/ecma262/。要到达这个最后阶段，该特性必须在 TC39 的 test262 测试套件[1]中进行验收测试；且至少要有两个通过测试的兼容实现(例如，Chrome Canary 的 V8 和 Firefox Nightly 的 SpiderMonkey，或 Firefox 的 SpiderMonkey 版本和 Safari 技术预览版的 JavaScriptCore 等)。满足这些条件后，该特性的开发团队将发送 PR(pull request)到 ecma262 仓库，并将规范的变化纳入编辑草案，ECMAScript 审校组将接受该 PR。至此，提案便进入阶段 4 了。

这就是提案成为 JavaScript 的一部分所要经历的流程，但并不是每个更改都是一个提案。相关人员在 TC39 会议上针对规范的 PR 达成的共识可实现较小的更改。例如，Date.prototype.toString 的输出在 ES2017 和 ES2018 之间发生的变化(请参见第 17 章)，就是相关人员对 PR(而不是经历不同阶段的提案)达成共识的结果。通常这些更改是编辑草案的更改，或者是 JavaScript 引擎已实现但未包含在规范中的更改，也可能是规范要求的并被 TC39 认为可取而且做到了“Web 兼容”(不会破坏大量现有代码)的改动，例如 ES2019 中的更改，它使 Array.prototype.sort 成为稳定的排序(请参见第 11 章)。如果你想了解哪些更改计划或者已经以这种方式进行，请查看 https://github.com/tc39/ecma262 仓库(同样，对于 ECMA-402 的更改，请查看 https://github.com/tc39/ecma402)中的“needs consensus”标签。要找到已完成的变更，请查看“has consensus”“editorial change”和“normative change”标签。在某些时候，这些以“needs consensus”为标签的更改可能会采用更正式的流程，但目前你可通过这种方式查看它们。

1.3.3 参与

如果你看到一个感兴趣的提案，想参与其中，那么你应该何时参与？参与的流程应该是怎样的？

1 https://github.com/tc39/test262。

重要的是尽早参与。提案进入阶段 3 后，相关人员通常仅考虑基于实现过程的重大问题。参与的最佳时机在阶段 0、阶段 1 和阶段 2。在这几个阶段，你可根据自己的经验提供见解，帮助定义语义，使用 Babel 之类的工具尝试提议的内容等。这并不是说你在阶段 3 的提案中起不到什么作用(有时候有一些任务，比如确定规范文本或者帮忙编写开发者文档)，只是要注意，在阶段 3 提出的修改建议通常没什么用，除非你是在 JavaScript 引擎中实现此提案的人之一。

如果你已经找到了一个你想参与的提案，该怎么办？这取决于你，但这里有一些建议。

- **充分调研。**请仔细阅读提案的说明(TC39 提案列表链接的 README.md)和其他相关文档。如果其中提到了现有技术(如另一种语言中的类似特性)，则应仔细阅读那项现有技术。如果有最初的规范文档，请阅读它(该指南可能会有帮助：https://timothygu.me/es-howto/)。请确保你提供的意见是有根据的。
- **试用该特性！**即使你还不能使用它，也可以写一些推测性代码(你无法运行，但可以思考的代码)来评估该提案在多大程度上能解决它计划要解决的问题。如果有相应的 Babel 插件，可尝试编写并运行代码。查看该特性的执行结果并提供反馈。
- **寻找你可以参与的方式。**除了建议和反馈之外，还有很多方法可让你参与到提案的改进中。例如，你可以寻找那些已达成共识，但没有人有时间去解决的问题(研究已存在的技术，更新说明文档和规范文档)。如果你愿意，可以做这些事情。你可通过 GitHub 的 issue 讨论提案，从而与提案作者协调。

参与过程中，请尊重每个人，并保持友好、耐心、包容和体贴的心态。要注意措辞。与其表现出反对、轻视或阻挠的态度，不如善待他人并表现出合作精神，这样，你更有可能产生影响。请注意，TC39 的会议和用于制定提案的线上空间均受《行为准则》[1]约束。请将不当行为提交给 TC39 的行为准则委员会(见链接文档)。

1.3.4　跟上新特性的步伐

事实上，你不必紧跟 JavaScript 的新特性，因为如前所述，之前的语法不会消失。但是，如果你正在阅读本书，那么我猜你一定想跟上新特性的步伐。

在阅读前面有关 TC39 流程的内容时，你可能感到困惑：该流程保证了新特性在被添加到规范中之前就能使用。然而，2009 年问世的 ES5 和 2015 年问世的 ES2015 描述的大部分特性在当时的任何 JavaScript 引擎中都不存在。如果规范的确定在新特性出来之后，那如何才能紧跟新的特性呢？下面给出了一些方法。

- 关注 GitHub 上的提案仓库(https://github.com/tc39/proposals)。如果提案进入阶段 3，则可能会在一两年内被添加到规范中。即使是处于阶段 2 的特性，最后也有可能被添加到规范中，不过任何特性都可能在任何阶段被 TC39 拒绝。
- 阅读发布在 https://github.com/tc39/notes 上的 TC39 的会议记录。
- 参加 TC39 论坛小组(https://es.discourse.group/)[2]。
- 注意下一节中讨论的工具的使用情况。

站点 https://thenewtoys.dev 将持续更新即将出现的新特性，敬请关注。

1 https://tc39.es/code-of-conduct/。

2 讨论小组在很大程度上取代了非正式讨论邮件列表。该列表仍然存在，但 TC39 的许多代表建议避免使用。

1.4 旧环境中使用新特性

如果仅学习本书中涵盖的特性，你不必担心如何处理不支持这些特性的环境。当前最新版本的 Chrome、Firefox、Safari、Edge 桌面浏览器以及 Node.js 几乎支持本书的所有内容(第 18 和 19 章除外)。我们使用其中之一运行代码即可。

使用 Node.js 运行示例

默认情况下，当你使用 Node.js 运行脚本时，如下所示:

```
node script.js
```

代码在模块作用域(而不是全局作用域)内运行。本书中一些示例的演示仅在全局作用域内生效，因此当你以这种方式运行代码时，它们将不起作用。

对于这些示例，请使用浏览器运行代码，或使用 Node.js 的交互式解释器(REPL)。要使用 REPL，不必为 node 命令的运行参数指定脚本文件，而是使用<运算符将脚本文件重定向到其中(这在 UNIX/Linux/Mac OS 系统和 Windows 上均有效)。

```
node < script.js
```

在此要提醒的是，需要在全局作用域内运行示例时再执行此操作。

不过，在某个阶段，你可能想在不支持新特性的环境中使用它们。例如，大多数 JavaScript 开发工作仍基于 Web 浏览器，不同浏览器中的 JavaScript 引擎对新特性的支持情况不同(Internet Explorer 甚至不支持本书讨论的任何新特性[1]，但目前它仍然占据了一部分全球市场份额，尤其是在政府和大型公司内部网中)。

当 ES5 发布时，确实存在这个问题，因为只有很少一部分新特性在当时发布的 JavaScript 引擎中得以支持。但是 ES5 的大多数特性都是新的标准库特性，而在语法上没有显著的更改，因此可使用 es5-shim.js[2]、core-js[3]、es-shims[4]等各种项目来 polyfill(引入一个额外的脚本来提供缺少的对象/函数)。不过，在 2010—2015 年的 ES2015 开发过程中，很明显，要做好新语法的开发工作，就必须具备新语法的实际开发经验，但是 JavaScript 的实现尚没有支持新语法，这显然是自相矛盾的。

工具制造者解决了这个问题！他们创建了 Traceur[5]和 Babel[6](以前被称为 6to5)等工具，这些工具以使用新语法的源代码作为输入，将其转换为使用旧语法的代码，然后输出旧语法风格的代码(还可选择使用 polyfills 和其他运行时支持方法)。类似地，TypeScript[7]在规范完成之前就支持 ES2015 的主要部分。这些工具使你可以编写新语法风格的代码，但在将其交付到旧环境之前需要将其转换为旧语法风格的代码。这种转换过程被称为“编译”或“转译”。这最初是为了方便对 ES2015 计划中的 JavaScript 改进进行反馈，但即使在 ES2015 出来以后，如果你计划在不支持新特性的环境中运行新语法风格的代码，它也是一种编写新风格代码的有用方法。

1 至少支持不充分，它有一个不完整的 let 和 const 版本。

2 https://github.com/es-shims/es5-shim。

3 https://github.com/zloirock/core-js。

4 https://github.com/es-shims/。

5 https://github.com/google/traceur-compiler。

6 http://babeljs.io/。

7 http://typescriptlang.org/。

在我撰写本书时，Traceur 已经销声匿迹了，但 Babel 却被全球很大一部分的 JavaScript 开发者所使用。Babel 几乎对 TC39 流程中的所有特性都进行了转换，甚至包括那些处于阶段 1 的特性，这些特性在进入下一个流程前可能会有明显的变化(因此，使用这些特性时需要自担风险。阶段 3 以上的特性要相对安全些)。选择你想要使用的转换插件，使用这些特性编写代码，Babel 会生成可在不支持这些特性的环境中使用的代码。

用 Babel 编译的示例

本节将简单介绍如何通过 Babel 将使用 ES2015 特性(箭头函数)的代码编译成可在 IE11 上运行的兼容 ES5 的代码。但这只是一个例子，你也可轻松地通过 Babel 将使用阶段 3 特性的代码转换为兼容 ES2020 的代码，而这些特性尚未出现在任何已发布的 JavaScript 引擎中。Babel 甚至还支持那些根本不在 TC39 流程中的特性转换，例如 JSX[1](在某些 JavaScript 框架中使用，主要是 React[2])。富有冒险精神的人可编写自己的转换插件，只用于他们的项目！

安装 Babel 前，你需要有 Node.js 和 npm(Node Package Manager)。如果你尚未在系统上安装它们，请执行以下任一操作。

(1) 在官网 https://nodejs.org/上查找适合你系统的安装程序/软件包并安装。

(2) 使用 NVM(Node Version Manager)，该工具提供了一种方便的方法来安装多个 Node 版本并在它们之间进行切换：https://github.com/nvm-sh/nvm。

npm 与 Node.js 捆绑在一起，因此不必单独安装它。

安装完成后，请执行以下操作。

(1) 为此示例创建一个目录(例如，在你的主目录下创建 example 目录)。

(2) 打开命令提示符/终端窗口并切换到刚才创建的目录。

(3) 使用 npm 创建 package.json 文件，输入：

```
npm init
```

然后按回车键。npm 会问一系列问题，根据实际情况回答对应的问题(或者只按回车键来回答所有问题)。完成后，它会在你的 example 目录下写入 package.json 文件。

(4) 下一步，安装 Babel(访问 https://babeljs.io/docs/setup/#installation，然后单击 CLI 按钮；可在这个网页上查看是否有更新)。输入：

```
npm install --save-dev @babel/core @babel/cli
```

然后按回车键。npm 将下载 Babel、命令行接口及其所有依赖项并将它们安装到示例项目中。(你可能会收到与 fsevents 模块相关的警告或一些 deprecation 警告，这没有关系。)

(5) 现在，你可通过直接输入 Babel 来使用它，但是我们可在 package.json 文件中添加一个 npm 脚本命令来简化对它的使用。用你最喜欢的编辑器打开 package.json 文件。如果没有顶层的 script 字段，请创建一个。(当前最新版本的 npm 会包含一个有 test 命令的 script 字段，打印一段错误信息。)在 script 字段中，添加以下设置：

```
"build": "babel src -d lib"
```

现在 package.json 文件的内容应该如代码清单 1-1 所示。在你的文件中，script 字段中可能还有一

1 https://facebook.github.io/jsx/。

2 https://reactjs.org/。

个 test 命令，这没关系。脚本也可能有不同的许可证，我总是将默认值改为 MIT。最后务必保存该文件。

代码清单 1-1　示例 package.json—— package.json

```
{
    "name": "example",
    "version": "1.0.0",
    "description": "",
    "main": "index.js",
    "scripts": {
       "build": "babel src -d lib"
    },
    "author": "",
    "license": "MIT",
    "devDependencies": {
      "@babel/cli": "^7.2.3",
      "@babel/core": "^7.2.2"
    }
}
```

(6) Babel 是高度模块化的。尽管我们已经安装了它，但还没有告诉它做任何事情。在此示例中，我们将使用它的一个预设，通过安装和配置该预设来告诉它将 ES2015 代码转换为 ES5 代码。要安装这个预设，请键入：

```
npm install --save-dev babel-preset-env
```

然后按回车键。在下一步中，我们将对其进行配置。

(7) 现在，需要为 Babel 创建一个配置文件.babelrc(注意前面的点号)。使用以下内容创建该文件(或使用章节下载中的文件)：

```
{
  "presets": [
    [
      "env",
      {
        "targets": {
          "ie": "11"
        }
      }
    ]
  ]
}
```

该配置告诉 Babel 使用其 env 预设，Babel 文档将其描述为“. . . a smart preset that allows you to use the latest JavaScript without needing to micromanage which syntax transforms . . . are needed by your target environment(s).(……一个智能预设，它允许你使用最新的 JavaScript 特性而不必单独管理需要语法转换的目标环境……)”。在这个配置中，如将目标设置为"ie":"11"，将告知 env 预设，你的目标环境是 IE11，这适用于下面的示例。在实际使用中，你需要根据自己的情况查看 env 预设[1]的文档或其他预设或插件的文档。

至此，本示例的 Babel 配置已经完成了。现在创建一些代码进行编译。在 example 目录中创建一

1 https://babeljs.io/docs/en/babel-preset-env#docsNav。

个名为 src 的子目录，并在其中创建一个名为 index.js 的文件，其内容如代码清单 1-2 所示。在此过程的最后，我将向你展示文件树最终的列表，因此，如果你不太确定文件是否创建在正确的目录下，请不必过于担心。只需要创建文件，如果它的位置不对，将其移到正确的位置即可。

代码清单 1-2　编译 ES2015 示例的输入——index.js

```
var obj = {
    rex: /\d/,
    checkArray: function(array) {
         return array.some(entry => this.rex.test(entry));
    }
};
console.log(obj.checkArray(["no", "digits", "in", "this", "array"])); // false
console.log(obj.checkArray(["this", "array", "has", "1", "digit"])); // true
```

代码清单 1-2 仅使用了 ES2015+的一项特性——箭头函数，即 some 函数调用中的 entry => this.rex.test (entry)——上面的代码中加粗显示的部分(是的，这确实是一个函数)。第 3 章将介绍该函数。如你所见，它通过简短、不完整的形式提供了一种定义函数的简洁方式，并且像封装变量一样对 this 进行了封装(而不是以调用它们的方式进行 this 设置)。当调用 obj.checkArray(...)时，即使在 some 回调方法中，该回调中的 this 变量指的也是 obj 对象，因此 this.rex 指的是 obj 对象的 rex 属性。如果这里的回调采用的是传统函数，那就不是这样了。

此时，example 目录的内容应如下所示：

```
example/
+-- node_modules/
|   +-- (various directories and files)
+-- src/
|   +-- index.js
+-- .babelrc
+-- package.json
+-- package-lock.json
```

准备好进行编译！请输入：

```
npm run build
```

然后按回车键。Babel 将进行它的工作，为你创建 lib 输出目录，并将 ES5 版本的 index.js 写入其中。最终的 lib / index.js 文件如代码清单 1-3 所示。

代码清单 1-3　编译 ES2015 示例的输出——index-transpiled-to-es5.js

```
"use strict";

var obj = {
  rex: /\d/,
  checkArray: function checkArray(array) {
    var _this = this;

    return array.some(function (entry) {
       return _this.rex.test(entry);
    });
  }
```

```
};
console.log(obj.checkArray(["no", "digits", "in", "this", "array"])); // false

console.log(obj.checkArray(["this", "array", "has", "1", "digit"])); // true
```

如果将 src / index.js(代码清单 1-2)与 lib / index.js(代码清单 1-3)进行比较，你会发现其中只有几处变化(空格除外)。首先，Babel 在编译后的文件顶部添加了"use strict";指令(回想一下，严格模式是 ES5 中添加的一个特性，它修改一些由于各种原因而存在问题的行为)。这是 Babel 的默认设置，但如果你的代码依赖于宽松模式，也可将其关闭。

但有趣的地方在于它重写箭头函数的方式。它在 checkArray 中创建了一个名为_this 的变量，将其值设置为 this，然后以传统函数作为 some 函数的回调；在函数中它用_this 代替 this。这与前面对箭头函数的描述相吻合—— 像封装变量一样对 this 进行了封装。Babel 只是以 ES5 环境可理解的方式实现了这一点。

显然，这只是一个很小的示例，但是它说明了如何在旧环境中使用新特性，并让你了解了在项目中需要这样做时可能使用的一个工具。无论是使用 Gulp[1]、Grunt[2]、Webpack[3]、Browserify[4]、Rollup[5]，还是其他任何工具，都可将 Babel 集成到你的构建系统中；https://babeljs.io/docs/setup/#installation 的安装页面提供了所有主要工具的说明。

1.5 本章小结

在过去的几年中，尤其是从 2015 年开始，JavaScript 发生了巨大的变化，未来它将继续变化。但是，不必担心所有新特性带来的学习负担。JavaScript 将始终保持向后兼容性，因此，除非你准备好并有需要，否则不必采用新特性。

新特性涉及的范围很广，涵盖小的调整(如允许函数调用参数列表中的尾后逗号和标准库中新的便捷方法)以及大的改进，如声明式类语法(第 4 章)、异步函数(第 9 章)和模块(第 13 章)。

最终负责推动 JavaScript 向前发展的是 TC39 技术委员会(Ecma International 的 Technical Committee 39)，该委员会的成员包括：JavaScript 开发者、编程语言研究人员、库和大型网站作者，主要 JavaScript 引擎的代表以及其他与 JavaScript 的成功与未来相关的人。但是任何人都可参与其中。

新特性的设计过程是公开和开放的。你可通过 GitHub 仓库、已发布的会议记录以及 TC39 论坛组来关注进度并获取最新消息。另外，本书的网站 https://thenewtoys.dev 将继续更新本书中没有覆盖的新特性。

本书介绍的大部分特性都已被 Chrome、Firefox、Safari 和 Edge 等主流浏览器中的 JavaScript 引擎以及非浏览器环境(如 Node.js、Electron、React Native 等)的引擎支持。

可通过 JavaScript-to-JavaScript 编译(也被称为“转译”)支持 Internet Explorer 等旧环境，使用可集成到构建系统中的工具(如 Babel)将新风格的代码转换为旧风格的代码。某些特性(如第 14 章介绍的 Proxy 对象)无法以这种方式得到完全支持，但大部分是可以的。

希望本章为你了解新特性打下了基础。

接下来的章节将介绍新特性！

1 https://gulpjs.com/。
2 https://gruntjs.com/。
3 https://webpack.js.org/。
4 http://browserify.org/。
5 https://rollupjs.org/。