High Performance JavaScript
Build Faster Web Application Interfaces

高性能 JavaScript

[美] *Nicholas C. Zakas* 著
丁琛 译 赵泽欣 审校

O'REILLY®
YAHOO! PRESS
中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

高性能 JavaScript

[美] *Nicholas C. Zakas* 著
丁琛 译 赵泽欣 审校

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

内 容 简 介

如果你使用 JavaScript 构建交互丰富的 Web 应用，那么 JavaScript 代码可能是造成你的 Web 应用速度变慢的主要原因。本书揭示的技术和策略能帮助你在开发过程中消除性能瓶颈。你将会了解如何提升各方面的性能，包括代码的加载、运行、DOM 交互、页面生存周期等。雅虎的前端工程师 Nicholas C. Zakas 和其他五位 JavaScript 专家介绍了页面代码加载的最佳方法和编程技巧，来帮助你编写更为高效和快速的代码。你还会了解到构建和部署文件到生产环境的最佳实践，以及有助于定位线上问题的工具。

版权贸易合同登记号　图字：01-2010-4394

图书在版编目（CIP）数据

高性能 JavaScript /（美）泽卡斯（Zakas,N.C.）著；丁琛译. —北京：电子工业出版社，2015.8
书名原文：High Performance JavaScript
ISBN 978-7-121-26677-5

Ⅰ. ①高… Ⅱ. ①泽… ②丁… Ⅲ. ①JAVA 语言－程序设计 Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字(2015)第 162448 号

策划编辑：张春雨
责任编辑：徐津平
封面设计：Karen Montgomery　张　健
印　　刷：三河市鑫金马印装有限公司
装　　订：三河市鑫金马印装有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编：100036
开　　本：787×980　1/16　印张：14.5　字数：344 千字
版　　次：2015 年 8 月第 1 版
印　　次：2015 年 8 月第 1 次印刷
定　　价：65.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。
质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。
服务热线：（010）88258888。

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

谨以此书献给我的家人，妈妈、爸爸和Grey，
是你们的爱和支持使我走过这些岁月。

译者序

这是一本关于 JavaScript 性能的书。

在 Web 应用日趋丰富的今天，越来越多的 JavaScript 被运用在我们的网页中。随着用户体验被日益重视，前端性能对用户体验的影响开始备受关注，而引起性能问题的因素相对复杂，因此它很难得到全面的解决。这本书是一个契机，它尝试着从多个方面综合分析导致性能问题的原因，并给出适合的解决方案，帮助我们改善 Web 应用的品质。

这本书页数不多，但它承载着 JavaScript 性能方面最为宝贵的经验。不仅从语言特性、数据结构、浏览器机理、网络传输等层面分析导致性能问题的原因，还介绍了多种工具来帮助我们提升开发过程和部署环节的工作效率。

本书作者 Nicholas C. Zakas 是一位经验丰富的前端专家，他的许多研究（*www.nczonline.net*）对前端业界的贡献让我们受益匪浅。本书的另外五位特约作者均为各自领域的专家，他们的专业技能和知识的融入使得本书内容更为充实，更具实用价值。

特别感谢赵泽欣（小马），他为审阅译文花了大量的时间和精力，他的耐心和细致让我十分敬佩。感谢朱宁（白鸦）和周筠老师的引荐让我得以参与本书的翻译。还要感谢博文视点的编辑们在本书翻译过程中给予的极大理解和帮助。

我们在本书翻译过程中力求保持行文流畅，但纰漏在所难免，恳请广大读者批评指正。关于本书的任何意见或想法，欢迎发送邮件至 hpj.feedback@gmail.com。

最后，希望本书能帮助业界同仁打造出性能更为卓越的 Web 产品。

丁琛

前言
Preface

当 JavaScript 作为 Netscape Navigator 浏览器的一部分在 1996 年首次出现时，性能问题并不重要。当时的互联网仍处在发展初期，各个方面都很慢。从拨号上网到低配置的家用电脑，上网冲浪往往比任何事情都需要耐心。人们都做好了等待网页加载的心理准备，页面加载能完成就是一件值得庆祝的事了。

JavaScript 最初的目标是改善网页的用户体验。JavaScript 能代替服务器处理页面中类似表单验证的简单任务，这样做节省了与服务器连接的大量时间。想象一下当你填完一个很长的表单，提交后等待了 30~60 秒，却得到一个字段出错的信息是什么感觉。显而易见，JavaScript 为早期的互联网用户节省了很多时间。

互联网的发展
The Internet Evolves

在接下来的 10 年里，电脑和互联网不断发展。首先，两者都变得更快。高速的微处理器，内存的廉价供应，以及光纤连接的出现将互联网推向了一个新的时代。随着高速网络的普及，网页开始变得丰富，并且承载着更多的信息和多媒体内容。Web 从简单的关联文档演变成了各式各样的设计和界面。一切都变了，除了一样东西，那就是 JavaScript。

这项曾用于节省服务器消耗的技术日益普及，但代码也从数十行的发展到成百上千行。IE 4 和动态 HTML [译注1]（改变页面显示而无需重新加载的技术）的推出更是使得网页中的 JavaScript 代码量只增不减。

最近一次浏览器的重大更新是文档对象模型（DOM）的推出，这是一个被 IE 5、Netscape 6 及 Opera 所一致接受的动态HTML 接口。紧接着，JavaScript 被标准化，并推出了 ECMA-262

译注1：Dynamic HTML（简称 DHTML）是一种通过结合 HTML/JavaScript/CSS 与 DOM，来创建动态网页内容的方法。它的历史发展和更多信息请参见 *http://en.wikipedia.org/wiki/Dynamic_HTML*。

第三版。随着所有浏览器都支持 DOM，同时（或多或少）提供了对相同版本 JavaScript 的支持，Web 应用平台诞生了。尽管有如此巨大的飞跃，有了编写 JavaScript 的通用 API，但是负责执行代码的 JavaScript 引擎几乎没有变化。

为什么优化是必要的
Why Optimization Is Necessary

在 1996 年，JavaScript 引擎只要能支持页面里数十行的 JavaScript 代码就好，而今天，却运行着成千上万行 JavaScript 代码的 Web 应用。从许多方面来说，如果不是因为浏览器自身在语言管理和基础设施方面的落后，JavaScript 本可能取得更大规模的成功。IE 6 就是一个明证，发布之初，它的稳定性和性能都被人们称颂，但后来却因为自身的 Bug 和反应迟钝而被痛批为令人讨厌的 Web 应用平台。

事实上，IE 6 并没有变慢，它只是被寄予了厚望。2001 年 IE 6 刚发布时出现的各类早期 Web 应用比 2005 年后出现的应用更轻量，JavaScript 代码也远没有那么多。JavaScript 代码数量的增长带来的影响变得明显，IE 6 的 JavaScript 引擎吃不消了，原因在于它的“静态垃圾回收机制”[译注2]。该引擎监视内存中固定数量的对象来确定何时进行垃圾回收。早期的 Web 应用开发人员很少会遇到这个极限值，随着更多的 JavaScript 代码产生越来越多的对象，复杂的 Web 应用开始频繁遭遇这个门槛。问题变得清晰起来：JavaScript 开发人员和 Web 应用都在发展，而 JavaScript 引擎却没有。

尽管其他浏览器有着更加完善的垃圾回收机制和更好的运行性能，但大多数仍然使用 JavaScript 解释器来执行代码。解释性代码天生就没有编译性代码快，因为解释性代码必须经历把代码转化成计算机指令的过程。无论解释器怎样优化和多么智能，它总是会带来一些性能损耗。

编译器已经有了各种各样的优化，使得开发人员可以按照他们想要的方式编写代码，而不需要担心是否是最优。编译器可以基于词法分析去判断代码想实现什么，然后产生出能完成任务的运行最快的机器码来进行优化。解释器很少有这样的优化，这很大程度上意味着，代码怎么写，就被怎么执行。

实际上，通常在其他语言中由编译器处理的优化，在 JavaScript 中却要求开发人员来完成。

译注2：“Static Garbage Collection”的详细解释请参见 *http://joses.st.ewi.tudelft.nl/static-gc.html*。关于 IE 6 的垃圾回收机制原理，请参见 MSDN 的文章 *http://blogs.msdn.com/ericlippert/archive/2003/09/17/53038.aspx*。

下一代 JavaScript 引擎
Next-Generation JavaScript Engines

2008 年，JavaScript 引擎迎来了第一次大的性能升级。Google 发布了全新的浏览器，名为 Chrome。Chrome 是第一款采用优化后的 JavaScript 引擎的浏览器，该引擎的研发代号为 V8。V8 是一款为 JavaScript 打造的实时（JIT）编译引擎，它把 JavaScript 代码转化为机器码来执行，所以给人的感觉是 JavaScript 的执行速度超快。

其他浏览器紧跟着也优化了它们的 JavaScript 引擎。Safari 4 发布了名为 SquirrelFish Extreme（或称为 Nitro）的 JIT JavaScript 引擎，而 Firefox 3.5 的 TraceMonkey 引擎对频繁执行的代码路径做了优化[译注3]。

这些全新的 JavaScript 引擎带来的是编译器层面的优化，这也是它应该做的。或许有一天，开发人员完全无须关心代码的性能优化。可是，那一天还未到来。

性能依然需要关注
Performance Is Still a Concern

尽管核心 JavaScript 的执行速度已经有所提高，但 JavaScript 仍然有多个方面的问题在新的引擎中没有被处理。网络延迟导致的滞缓和影响页面外观的操作尚未得到浏览器的充分优化。尽管诸如函数内联、代码合并以及字符串连接算法等简单的优化很容易通过编译器进行优化，但对动态且多层结构的 Web 应用程序来说，这些优化只能解决部分的性能问题。

然而全新的 JavaScript 引擎让我们似乎看到未来高速互联网的模样，在可预见到的未来，当今的性能话题仍然具有相关性和重要性。

本书中讨论的技术和方案涉及 JavaScript 的各个方面，内容涵盖运行时间、下载、DOM 操作、页面生存周期等。这些话题仅仅是一小部分，它们相关的核心（ECMAScript）性能可能随着 JavaScript 的不断进步而变得无关紧要，但这还有待时日。

其他的话题则针对那些更快的 JavaScript 引擎也力不能及的方面：DOM 交互、网络延迟、JavaScript 的阻塞和并发下载等。这些话题在未来依然重要，而且还会再受到更大关注，它们需要从底层研究 JavaScript 执行时间才能继续提高。

译注3：截止到译者翻译本书之时，Firefox 4.0 版本已经推出 beta 版本，并宣称启用了性能更优的 JaegerMonkey 引擎。关于浏览器脚本引擎的性能演化及更多其他信息，请参见 *http://zh.wikipedia.org/zh-cn/JavaScript* 引擎。

本书的组织
How This Book Is Organized

本书章节的组织方式基于一个正常的 JavaScript 开发周期。最开始的第 1 章讨论页面加载 JavaScript 的最佳方式。第 2 章到第 8 章专注于那些有助于你的 JavaScript 代码尽可能高效运行的编程技术。第 9 章讨论构建和部署 JavaScript 文件到生产环境的最佳方法，第 10 章提到的性能工具能帮助你找出代码部署后的潜在问题。以下 5 章是由特约作者完成的：

- 第 3 章　DOM 编程，由 Stoyan Stefanov 完成。
- 第 5 章　字符串和正则表达式，由 Steven Levithan 完成。
- 第 7 章　Ajax，由 Ross Harmes 完成。
- 第 9 章　构建并部署高性能 JavaScript 应用，由 Julien Lecomte 完成。
- 第 10 章　工具，由 Matt Sweeney 完成。

这些作者都是对 Web 开发社区有着重要贡献的资深 Web 开发人员。为便于你识别这些章节，他们的名字会出现在各自章节的起始页。

JavaScript 加载
JavaScript Loading

第 1 章，“加载和执行”，从 JavaScript 的基础开始：把代码加载到页面。要提高 JavaScript 的性能，首先要用最高效的方式加载代码到页面中。本章专注于与加载 JavaScript 代码相关的性能问题，并给出了几种方法以减轻它带来的负面影响。

编码技术
Coding Technique

JavaScript 中大量性能问题的来源是因为使用低效的算法或工具编写出的糟糕代码。接下来的 7 个章节专注于找出问题代码并给出更快替代方案来完成相同的任务。

第 2 章“数据访问”，重点介绍了 JavaScript 如何存取脚本里的数据。数据的储存位置同数据类型同样重要，本章解释了作用域链和原型链对脚本整体性能的影响。

Stoyan Stefanov，他对 Web 浏览器的内部工作机制十分在行，所以由他编写第 3 章“DOM 编程”。Stoyan 解释了在 JavaScript 中 DOM 交互比其他类型的操作要慢的原因是因为它的实现机制。他论述了 DOM 相关的所有内容，包括描述重绘和重排是如何拖慢你的代码的。

第 4 章“算法和流控制”，解释了常用编程模式（比如循环和迭代）是如何影响运行期性能的。本章还讨论了诸如 memoization 的优化技术以及浏览器的 JavaScript 运行期限制。

许多 Web 应用程序都会执行复杂的的 JavaScript 字符串操作，字符串专家 Steven Levithan 在第 5 章“字符串和正则表达式”全面介绍了相关主题。Web 开发人员已经与浏览器低效的字符串处理奋战多年，Steven 会解释为什么一些字符串操作起来会慢，以及如何应对。

第 6 章“快速响应的用户界面”，聚焦于用户体验。JavaScript 可能会导致浏览器假死，让用户感到无比沮丧。本章讨论了多种技术来确保用户界面总是处于可快速响应的状态。

第 7 章“Ajax”，Ross Harmes 讨论了实现客户端-服务端快速通信的最佳方法。 Ross 还提到数据格式是如何影响 Ajax 性能的以及为什么 `XMLHttpRequest` 并非总是最佳选择。

第 8 章“编程实践”介绍了一些专门针对 JavaScript 的最佳实践。

部署
Deployment

JavaScript 代码一旦编写完成并通过测试，就是时候向用户发布了。然而，你不应当直接把源码放到生产环境。Julien Lecomte 在第 9 章“构建和部署高性能 JavaScript 应用”中介绍了如何使用一个构建系统去自动压缩文件，并使用 HTTP 压缩传输内容到浏览器。

测试
Testing

当所有 JavaScript 代码部署完成，下一步就要开始性能测试。Matt Sweeney 在第 10 章“工具”里的内容涵盖了测试方法论和相关工具。他介绍了如何使用 JavaScript 来衡量性能，还描述了两类通用工具，一类用于评估 JavaScript 运行期性能，另一类通过使用 HTTP 嗅探来发现隐藏的性能问题。

本书读者
Who This Book Is For

本书面向有中高级 JavaScript 经验且希望提升 Web 应用性能的 Web 开发人员。

本书的约定
Conventions Used in This Book

本书使用下列排版约定：

斜体（*Italic*）

表示专业词汇、链接（URL）、文件名和文件扩展名。

等宽字体（`Constant width`）

表示广义上的计算机代码，它们包括变量或函数名、数据库、数据类型、环境变量、语句和关键字。

这个图标表示提示、建议或一般说明。

这个图标表示警告或提醒。

代码用例
Using Code Examples

这本书是为了帮助你做好工作。一般来说，你可以在程序和文档中使用本书的代码。你无须联系我们获取许可。例如，使用来自本书的几段代码写一个程序是不需要许可的。出售和散布 O'Reilly 书中用例的光盘（CD-ROM）是需要许可的。通常引用本书用例和代码来回答问题是不需要许可的。把本书中大量的用例代码并入到你的产品文档中是需要许可的。

我们赞赏但不强求注明信息来源。一条信息来源通常包括标题、作者、出版者和国际标准书号（ISBN）。例如："High Performance JavaScript, by Nicholas C. Zakas. Copyright 2010 Yahoo!, Inc.,978-0-596-80279-0."。

如果你感到对示例代码的使用超出了正当引用或这里给出的许可范围，请随时通过 *permissions@oreilly.com* 联系我们。

意见和问题

Comments and Questions

请将对本书的评价和存在的问题通过如下地址告知出版者。

美国：

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

北京市西城区西直门南大街2号成铭大厦C座807室（100035）
奥莱利技术咨询（北京）有限公司

对于本书的评论或技术性的问题，请发送电子邮件到：

bookquestions@oreilly.com

想了解关于 O'Reilly 图书、课程、会议和新闻的更多信息，请参阅我们的网站：

http://www.oreilly.com
http://www.oreilly.com.cn

与本书有关的在线信息如下所示：

http://www.oreilly.com/catalog/9780596802790（原书）

致谢

Acknowledgments

首先，也是最重要的，我要感谢本书的所有特约作者：Matt Sweeney、Stoyan Stefanov、Stephen Levithan、Ross Harmes 和 Julien Lecomte。他们的专业技能和知识的结合使得本书的编写过程让人兴奋，让本书更加值得期待。

感谢所有我有幸会面和交流的来自世界各地的性能专家，特别是 Steve Souders、Tenni Theurer 和 Nicole Sullivan。你们三位帮助我扩大了在 Web 性能方面的视野，我心存感激。

还要感谢每一位帮忙审阅本书的人们，包括 Ryan Grove、Oliver Hunt、Matthew Russell、Ted Roden、Remy Sharp 和 Venkateswaran Udayasankar。

还要特别感谢 O'Reilly 和 Yahoo! 的每一位成就此书的人。我在 2006 年刚刚加入 Yahoo! 时就希望为它写一本书，Yahoo! Press 让这件事成为现实。

目录

Table of contents