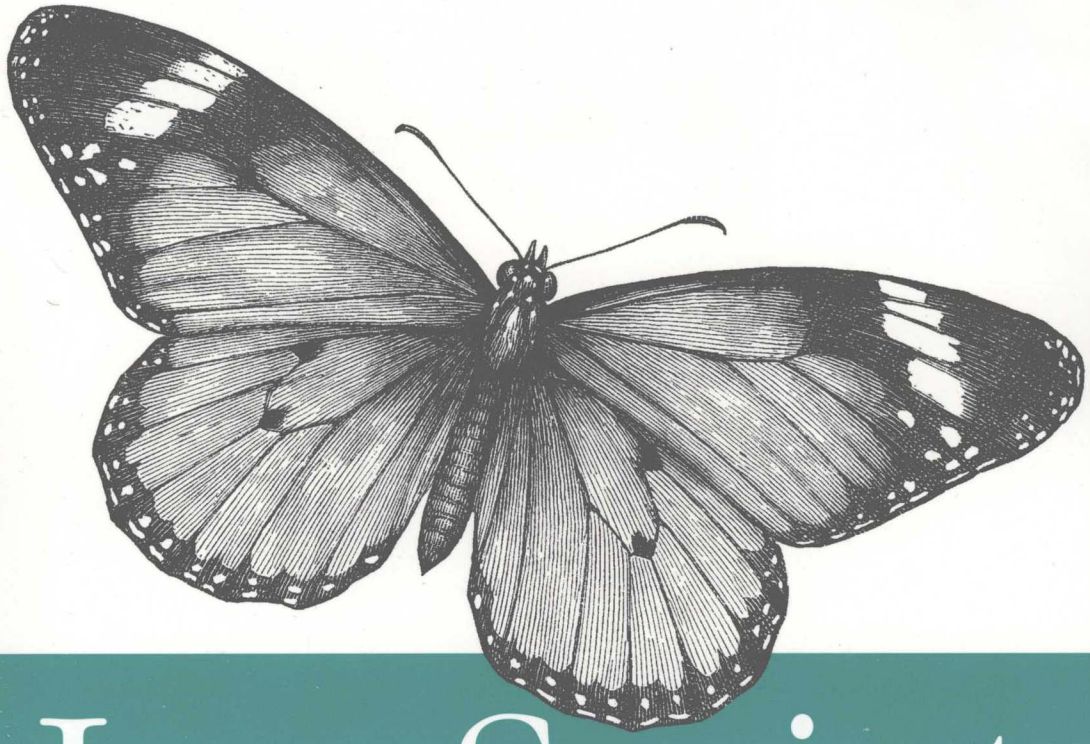

JavaScript: The Good Parts

O'REILLY® | YAHOO! PRESS
東南大學出版社

Douglas Crockford 著

JavaScript: The Good Parts（影印版）

JavaScript: The Good Parts

Douglas Crockford

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Taipei • Tokyo

O'Reilly Media, Inc. 授权东南大学出版社出版

南京 东南大学出版社

图书在版编目（CIP）数据

JavaScript:优良的部分 = JavaScript:The Good Parts:
英文/（美）克罗克福特（Crockford, D.）著. —影印本.
南京：东南大学出版社，2009.1
书名原文：JavaScript:The Good Parts
ISBN 978-7-5641-1447-3

Ⅰ.J… Ⅱ.克… Ⅲ.Java语言－程序设计－英文
Ⅳ.TP312

中国版本图书馆CIP数据核字（2008）第166986号

江苏省版权局著作权合同登记
图字：10-2008-346号

JavaScript:The Good Parts（影印版）

出版发行：东南大学出版社
地　　址：南京四牌楼2号　　邮编：210096
出 版 人：江　汉
网　　址：http://press.seu.edu.cn
电子邮件：press@seu.edu.cn
印　　刷：扬中市印刷有限公司
开　　本：787毫米×980毫米　16开本
印　　张：10.75印张
字　　数：181千字
版　　次：2009年1月第1版
印　　次：2009年1月第1次印刷
书　　号：ISBN 978-7-5641-1447-3/TP·239
印　　数：1~3000册
定　　价：28.00元（册）

本社图书若有印装质量问题，请直接与读者服务部联系。电话（传真）：025-83792328

O'Reilly Media, Inc.介绍

O'Reilly Media, Inc.是世界上在 UNIX、X、Internet 和其他开放系统图书领域具有领导地位的出版公司，同时是联机出版的先锋。

从最畅销的《The Whole Internet User's Guide & Catalog》（被纽约公共图书馆评为二十世纪最重要的50本书之一）到GNN（最早的Internet门户和商业网站），再到WebSite（第一个桌面PC的Web服务器软件），O'Reilly Media, Inc.一直处于Internet发展的最前沿。

许多书店的反馈表明，O'Reilly Media, Inc.是最稳定的计算机图书出版商 —— 每一本书都一版再版。与大多数计算机图书出版商相比，O'Reilly Media, Inc.具有深厚的计算机专业背景，这使得O'Reilly Media, Inc.形成了一个非常不同于其他出版商的出版方针。O'Reilly Media, Inc.所有的编辑人员以前都是程序员，或者是顶尖级的技术专家。O'Reilly Media, Inc.还有许多固定的作者群体 —— 他们本身是相关领域的技术专家、咨询专家，而现在编写著作，O'Reilly Media, Inc.依靠他们及时地推出图书。因为O'Reilly Media, Inc.紧密地与计算机业界联系着，所以O'Reilly Media, Inc.知道市场上真正需要什么图书。

出版说明

随着计算机技术的成熟和广泛应用，人类正在步入一个技术迅猛发展的新时期。计算机技术的发展给人们的工业生产、商业活动和日常生活都带来了巨大的影响。然而，计算机领域的技术更新速度之快也是众所周知的，为了帮助国内技术人员在第一时间了解国外最新的技术，东南大学出版社和美国 O'Reilly Meida, Inc.达成协议，将陆续引进该公司的代表前沿技术或者在某专项领域享有盛名的著作，以影印版或者简体中文版的形式呈献给读者。其中，影印版书籍力求与国外图书“同步”出版，并且“原汁原味”展现给读者。

我们真诚地希望，所引进的书籍能对国内相关行业的技术人员、科研机构的研究人员和高校师生的学习和工作有所帮助，对国内计算机技术的发展有所促进。也衷心期望读者提出宝贵的意见和建议。

最新出版的影印版图书，包括：

- 《JavaScript: The Good Parts》（影印版）
- 《学习 ActionScript 3.0》（影印版）
- 《深入浅出 C#》（影印版）
- 《深入浅出软件开发》（影印版）
- 《Ruby 程序设计语言》（影印版）
- 《高性能网站》（影印版）
- 《学习 ASP.NET 2.0 与 Ajax》（影印版）
- 《ASP.NET 3.5 构建 Web 2.0 门户网站》（影印版）
- 《Mac OS X: The Missing Manual, Leopard Editon》（影印版）
- 《深入浅出 JavaScript》（影印版）
- 《敏捷开发艺术》（影印版）
- 《Intermediate Perl》（影印版）
- 《学习 Python 第三版》（影印版）
- 《ASP.NET AJAX 编程》（影印版）

Preface

If we offend, it is with our good will
That you should think, we come not to offend,
But with good will. To show our simple skill,
That is the true beginning of our end.
—William Shakespeare, *A Midsummer Night's Dream*

This is a book about the JavaScript programming language. It is intended for programmers who, by happenstance or curiosity, are venturing into JavaScript for the first time. It is also intended for programmers who have been working with JavaScript at a novice level and are now ready for a more sophisticated relationship with the language. JavaScript is a surprisingly powerful language. Its unconventionality presents some challenges, but being a small language, it is easily mastered.

My goal here is to help you to learn to think in JavaScript. I will show you the components of the language and start you on the process of discovering the ways those components can be put together. This is not a reference book. It is not exhaustive about the language and its quirks. It doesn't contain everything you'll ever need to know. That stuff you can easily find online. Instead, this book just contains the things that are really important.

This is not a book for beginners. Someday I hope to write a *JavaScript: The First Parts* book, but this is not that book. This is not a book about Ajax or web programming. The focus is exclusively on JavaScript, which is just one of the languages the web developer must master.

This is not a book for dummies. This book is small, but it is dense. There is a lot of material packed into it. Don't be discouraged if it takes multiple readings to get it. Your efforts will be rewarded.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, filenames, and file extensions.

`Constant width`
: Indicates computer coding in a broad sense. This includes commands, options, variables, attributes, keys, requests, functions, methods, types, classes, modules, properties, parameters, values, objects, events, event handlers, XML and XHTML tags, macros, and keywords.

`Constant width bold`
: Indicates commands or other text that should be typed literally by the user.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*JavaScript: The Good Parts* by Douglas Crockford. Copyright 2008 Yahoo! Inc., 978-0-596-51774-8."

If you feel your use of code examples falls outside fair use or the permission given here, feel free to contact us at *permissions@oreilly.com*.

Safari® Books Online

When you see a Safari® Books Online icon on the cover of your favorite technology book, that means the book is available online through the O'Reilly Network Safari Bookshelf.

Safari offers a solution that's better than e-books. It's a virtual library that lets you easily search thousands of top tech books, cut and paste code samples, download chapters, and find quick answers when you need the most accurate, current information. Try it for free at *http://safari.oreilly.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

http://www.oreilly.com/catalog/9780596517748/

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our web site at:

http://www.oreilly.com/

Acknowledgments

I want to thank the reviewers who pointed out my many egregious errors. There are few things better in life than having really smart people point out your blunders. It is even better when they do it before you go public. Thank you, Steve Souders, Bill Scott, Julien Lecompte, Stoyan Stefanov, Eric Miraglia, and Elliotte Rusty Harold.

I want to thank the people I worked with at Electric Communities and State Software who helped me discover that deep down there was goodness in this language, especially Chip Morningstar, Randy Farmer, John La, Mark Miller, Scott Shattuck, and Bill Edney.

I want to thank Yahoo! Inc. for giving me time to work on this project and for being such a great place to work, and thanks to all members of the Ajax Strike Force, past and present. I also want to thank O'Reilly Media, Inc., particularly Mary Treseler, Simon St.Laurent, and Sumita Mukherji for making things go so smoothly.

Special thanks to Professor Lisa Drake for all those things she does. And I want to thank the guys in ECMA TC39 who are struggling to make ECMAScript a better language.

Finally, thanks to Brendan Eich, the world's most misunderstood programming language designer, without whom this book would not have been necessary.

About the Author

Douglas Crockford is a senior JavaScript architect at Yahoo! who is well known for discovering and popularizing the JSON (JavaScript Object Notation) format. He is the world's foremost living authority on JavaScript. He speaks regularly at conferences about advanced web technology, and he also serves on the ECMAScript committee.

Colophon

The animal on the cover of *JavaScript: The Good Parts* is a Plain Tiger butterfly (*Danaus chrysippus*). Outside of Asia, the insect is also known as the African Monarch. It is a medium-size butterfly characterized by bright orange wings with six black spots and alternating black-and-white stripes.

Its striking looks have been noted for millennia by scientists and artists. The writer Vladimir Nabokov—who was also a noted lepidopterist—had admiring words for the butterfly in an otherwise scathing *New York Times* book review of Alice Ford's *Audubon's Butterflies, Moths, and Other Studies* (The Studio Publications). In the book, Ford labels drawings made previous to and during Audubon's time in the 19th century as "scientifi-cally [*sic*] unsophisticated."

In response to Ford, Nabokov writes, "The unsophistication is all her own. She might have looked up John Abbot's prodigious representations of North American lepidoptera, 1797, or the splendid plates of 18th- and early-19th-century German lepidopterists. She might have traveled back some 33 centuries to the times of Tuthmosis IV or Amenophis III and, instead of the obvious scarab, found there frescoes with a marvelous Egyptian butterfly (subtly combining the pattern of our Painted Lady and the body of an African ally of the Monarch)."

While the Plain Tiger's beauty is part of its charm, its looks can also be deadly. During its larval stages, the butterfly ingests alkaloids that are poisonous to birds—its main predator—which are often attracted to the insect's markings. After ingesting the Plain Tiger, a bird will vomit repeatedly—occasionally fatally. If the bird lives, it will let other birds know to avoid the insect, which can also be recognized by its leisurely, meandering pattern of flying low to the earth.

The cover image is from *Dover's Animals*. The cover font is Adobe ITC Garamond. The text font is Linotype Birka; the heading font is Adobe Myriad Condensed; and the code font is LucasFont's TheSans Mono Condensed.

Table of Contents

CHAPTER 1
Good Parts

...setting the attractions of my
good parts aside I have no other charms.
—William Shakespeare, *The Merry Wives of Windsor*

When I was a young journeyman programmer, I would learn about every feature of the languages I was using, and I would attempt to use all of those features when I wrote. I suppose it was a way of showing off, and I suppose it worked because I was the guy you went to if you wanted to know how to use a particular feature.

Eventually I figured out that some of those features were more trouble than they were worth. Some of them were poorly specified, and so were more likely to cause portability problems. Some resulted in code that was difficult to read or modify. Some induced me to write in a manner that was too tricky and error-prone. And some of those features were design errors. Sometimes language designers make mistakes.

Most programming languages contain good parts and bad parts. I discovered that I could be a better programmer by using only the good parts and avoiding the bad parts. After all, how can you build something good out of bad parts?

It is rarely possible for standards committees to remove imperfections from a language because doing so would cause the breakage of all of the bad programs that depend on those bad parts. They are usually powerless to do anything except heap more features on top of the existing pile of imperfections. And the new features do not always interact harmoniously, thus producing more bad parts.

But *you* have the power to define your own subset. You can write better programs by relying exclusively on the good parts.

JavaScript is a language with more than its share of bad parts. It went from nonexistence to global adoption in an alarmingly short period of time. It never had an interval in the lab when it could be tried out and polished. It went straight into Netscape Navigator 2 just as it was, and it was very rough. When Java™ applets failed, JavaScript became the "Language of the Web" by default. JavaScript's popularity is almost completely independent of its qualities as a programming language.

Fortunately, JavaScript has some extraordinarily good parts. In JavaScript, there is a beautiful, elegant, highly expressive language that is buried under a steaming pile of good intentions and blunders. The best nature of JavaScript is so effectively hidden that for many years the prevailing opinion of JavaScript was that it was an unsightly, incompetent toy. My intention here is to expose the goodness in JavaScript, an outstanding, dynamic programming language. JavaScript is a block of marble, and I chip away the features that are not beautiful until the language's true nature reveals itself. I believe that the elegant subset I carved out is vastly superior to the language as a whole, being more reliable, readable, and maintainable.

This book will not attempt to fully describe the language. Instead, it will focus on the good parts with occasional warnings to avoid the bad. The subset that will be described here can be used to construct reliable, readable programs small and large. By focusing on just the good parts, we can reduce learning time, increase robustness, and save some trees.

Perhaps the greatest benefit of studying the good parts is that you can avoid the need to unlearn the bad parts. Unlearning bad patterns is very difficult. It is a painful task that most of us face with extreme reluctance. Sometimes languages are subsetted to make them work better for students. But in this case, I am subsetting JavaScript to make it work better for professionals.

Why JavaScript?

JavaScript is an important language because it is the language of the web browser. Its association with the browser makes it one of the most popular programming languages in the world. At the same time, it is one of the most despised programming languages in the world. The API of the browser, the Document Object Model (DOM) is quite awful, and JavaScript is unfairly blamed. The DOM would be painful to work with in any language. The DOM is poorly specified and inconsistently implemented. This book touches only very lightly on the DOM. I think writing a *Good Parts* book about the DOM would be extremely challenging.

JavaScript is most despised because it isn't SOME OTHER LANGUAGE. If you are good in SOME OTHER LANGUAGE and you have to program in an environment that only supports JavaScript, then you are forced to use JavaScript, and that is annoying. Most people in that situation don't even bother to learn JavaScript first, and then they are surprised when JavaScript turns out to have significant differences from the SOME OTHER LANGUAGE they would rather be using, and that those differences matter.

The amazing thing about JavaScript is that it is possible to get work done with it without knowing much about the language, or even knowing much about programming. It is a language with enormous expressive power. It is even better when you know what you're doing. Programming is difficult business. It should never be undertaken in ignorance.

Analyzing JavaScript

JavaScript is built on some very good ideas and a few very bad ones.

The very good ideas include functions, loose typing, dynamic objects, and an expressive object literal notation. The bad ideas include a programming model based on global variables.

JavaScript's functions are first class objects with (mostly) lexical scoping. JavaScript is the first lambda language to go mainstream. Deep down, JavaScript has more in common with Lisp and Scheme than with Java. It is Lisp in C's clothing. This makes JavaScript a remarkably powerful language.

The fashion in most programming languages today demands strong typing. The theory is that strong typing allows a compiler to detect a large class of errors at compile time. The sooner we can detect and repair errors, the less they cost us. JavaScript is a loosely typed language, so JavaScript compilers are unable to detect type errors. This can be alarming to people who are coming to JavaScript from strongly typed languages. But it turns out that strong typing does not eliminate the need for careful testing. And I have found in my work that the sorts of errors that strong type checking finds are not the errors I worry about. On the other hand, I find loose typing to be liberating. I don't need to form complex class hierarchies. And I never have to cast or wrestle with the type system to get the behavior that I want.

JavaScript has a very powerful object literal notation. Objects can be created simply by listing their components. This notation was the inspiration for JSON, the popular data interchange format. (There will be more about JSON in Appendix E.)

A controversial feature in JavaScript is prototypal inheritance. JavaScript has a class-free object system in which objects inherit properties directly from other objects. This is really powerful, but it is unfamiliar to classically trained programmers. If you attempt to apply classical design patterns directly to JavaScript, you will be frustrated. But if you learn to work with JavaScript's prototypal nature, your efforts will be rewarded.

JavaScript is much maligned for its choice of key ideas. For the most part, though, those choices were good, if unusual. But there was one choice that was particularly bad: JavaScript depends on global variables for linkage. All of the top-level variables of all compilation units are tossed together in a common namespace called *the global object*. This is a bad thing because global variables are evil, and in JavaScript they are fundamental. Fortunately, as we will see, JavaScript also gives us the tools to mitigate this problem.

In a few cases, we can't ignore the bad parts. There are some unavoidable awful parts, which will be called out as they occur. They will also be summarized in Appendix A. But we will succeed in avoiding most of the bad parts in this book, summarizing much of what was left out in Appendix B. If you want to learn more about the bad parts and how to use them badly, consult any other JavaScript book.

The standard that defines JavaScript (aka JScript) is the third edition of *The ECMAScript Programming Language*, which is available from *http://www.ecma-international.org/publications/files/ecma-st/ECMA-262.pdf*. The language described in this book is a proper subset of ECMAScript. This book does not describe the whole language because it leaves out the bad parts. The treatment here is not exhaustive. It avoids the edge cases. You should, too. There is danger and misery at the edges.

Appendix C describes a programming tool called JSLint, a JavaScript parser that can analyze a JavaScript program and report on the bad parts that it contains. JSLint provides a degree of rigor that is generally lacking in JavaScript development. It can give you confidence that your programs contain only the good parts.

JavaScript is a language of many contrasts. It contains many errors and sharp edges, so you might wonder, "Why should I use JavaScript?" There are two answers. The first is that you don't have a choice. The Web has become an important platform for application development, and JavaScript is the only language that is found in all browsers. It is unfortunate that Java failed in that environment; if it hadn't, there could be a choice for people desiring a strongly typed classical language. But Java did fail and JavaScript is flourishing, so there is evidence that JavaScript did something right.

The other answer is that, despite its deficiencies, *JavaScript is really good.* It is lightweight and expressive. And once you get the hang of it, functional programming is a lot of fun.

But in order to use the language well, you must be well informed about its limitations. I will pound on those with some brutality. Don't let that discourage you. The good parts are good enough to compensate for the bad parts.

A Simple Testing Ground

If you have a web browser and any text editor, you have everything you need to run JavaScript programs. First, make an HTML file with a name like *program.html*:

```
<html><body><pre><script src="program.js">
</script></pre></body></html>
```

Then, make a file in the same directory with a name like *program.js*:

```
document.writeln('Hello, world!');
```

Next, open your HTML file in your browser to see the result. Throughout the book, a `method` method is used to define new methods. This is its definition:

```
Function.prototype.method = function (name, func) {
    this.prototype[name] = func;
    return this;
};
```

It will be explained in Chapter 4.

CHAPTER 2
Grammar

I know it well:
I read it in the grammar long ago.
—William Shakespeare, *The Tragedy of Titus Andronicus*

This chapter introduces the grammar of the good parts of JavaScript, presenting a quick overview of how the language is structured. We will represent the grammar with railroad diagrams.

The rules for interpreting these diagrams are simple:

- You start on the left edge and follow the tracks to the right edge.
- As you go, you will encounter literals in ovals, and rules or descriptions in rectangles.
- Any sequence that can be made by following the tracks is legal.
- Any sequence that cannot be made by following the tracks is not legal.
- Railroad diagrams with one bar at each end allow whitespace to be inserted between any pair of tokens. Railroad diagrams with two bars at each end do not.

The grammar of the good parts presented in this chapter is significantly simpler than the grammar of the whole language.

Whitespace

Whitespace can take the form of formatting characters or comments. Whitespace is usually insignificant, but it is occasionally necessary to use whitespace to separate sequences of characters that would otherwise be combined into a single token. For example, in:

```
var that = this;
```

the space between var and that cannot be removed, but the other spaces can be removed.