Extracted from:

Async JavaScript

Build More Responsive Apps with Less Code

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Pragmatic exPress

Async JavaScript

Build More Responsive Apps with Less Code

Trevor Burnham

edited by Jacquelyn Carter

Async JavaScript

Build More Responsive Apps with Less Code

Trevor Burnham

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The team that produced this book includes:

Jacquelyn Carter (editor)
Kim Wimpsett (copyeditor)
David J Kelly (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)

Printed in the United States of America.
ISBN-13: 978-1-937785-27-7
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—November 2012

Introduction

Originally devised to enhance web pages in Netscape 2.0, JavaScript is now faced with being a single-threaded language in a multimedia, multitasking, multicore world. Yet JavaScript has not only persevered since 1995, it's thrived. One after the other, potential rivals in the browser—Flash, Silverlight, and Java applets, to name a few—have come and (more or less) gone.

Meanwhile, when a programmer named Ryan Dahl wanted to build a new framework for event-driven servers, he searched the far reaches of computer science for a language that was both dynamic and single-threaded before realizing that the answer was right in front of him. And so, Node.js was born, and JavaScript became a force to be reckoned with in the server world.

How did this happen? As recently as 2001, Paul Graham wrote the following in his essay "The Other Road Ahead":[1]

> I would not even use JavaScript, if I were you... Most of the JavaScript I see on the Web isn't necessary, and much of it breaks.

Today, Graham is the lead partner at Y Combinator, the investment group behind Dropbox, Heroku, and hundreds of other start-ups—nearly all of which use JavaScript. As he put it in a revised version of the essay, "JavaScript now works."

When did JavaScript become a respectable language? Some say the turning point was Gmail (2004), which showed the world that with a heavy dose of Ajax you could run a first-class email client in the browser. Others say that it was jQuery (2006), which abstracted the rival browser APIs of the time to create a *de facto* standard. (As of 2011, 48 percent of the top 17,000 websites use jQuery.[2])

1. A revised version of this essay can be found at http://paulgraham.com/road.html. The original footnote can be found in the book *Hackers & Painters*.
2. http://appendto.com/jquery-overtakes-flash

Whatever the reason, JavaScript is here to stay. Apple got behind JavaScript with WebKit and Safari. Microsoft is getting behind JavaScript with Metro. Even Adobe is getting behind JavaScript with tools to generate HTML5 instead of Flash. What began as a humble browser feature has become arguably the most important programming language in the world.

Thanks to the ubiquity of web browsers, JavaScript has come closer than any other language to fulfilling Java's old promise of "write once, run anywhere." In 2007, Jeff Atwood coined *Atwood's law:*

> Any application that can be written in JavaScript will eventually be written in JavaScript.[3]

Trouble in Paradise

JavaScript was conceived to be a single-threaded language where asynchronous tasks are handled with events. When there are only a few potential events, event-based code is much simpler than multithreaded code. It's conceptually elegant, and it eliminates the need to wrap up data in mutexes and semaphores to make it thread-safe. But when a number of events are expected, with state that needs to be carried from one event to the next, that simplicity often gives way to a code structure so terrifying that it's been dubbed the *Pyramid of Doom*.

```
step1(function(result1) {
  step2(function(result2) {
    step3(function(result3) {
      // and so on...
    });
  });
});
```

"I love async, but I can't code like this," one developer famously complained on the Node.js Google Group.[4] But the problem isn't with the language itself; it's with the way programmers use the language. Dealing with complex sets of events in an elegant way is still frontier territory in JavaScript.

So, let's push the frontier forward! Let's prove to the world that even the most complex problems can be tackled with clean, maintainable JavaScript code.

3. http://www.codinghorror.com/blog/2007/07/the-principle-of-least-power.html
4. https://groups.google.com/forum/#!topic/nodejs/wzSUdkPICWg

Who Is This Book For?

This book is aimed at intermediate JavaScripters. You should know how variables are scoped. Keywords like `typeof`, `arguments`, and `this` shouldn't faze you. Perhaps most importantly, you should understand that

```
func(function(arg) { return next(arg); });
```

is just a needlessly verbose way of writing

```
func(next);
```

except in rare cases. (See Reg Braithwaite's excellent article "Captain Obvious on JavaScript" for more examples of small but important functional idioms.)[5]

What you *don't* need to know is how asynchronous events are scheduled in JavaScript. We'll cover that in the next chapter.

Resources for Learning JavaScript

As JavaScript has become the *lingua franca* of the Web (not to mention mobile devices), a vast number of informative books, courses, and sites devoted to it have appeared. Here are a few that I recommend:

- If you're new to programming altogether, check out the interactive tutorial site Codecademy.[6]
- If you're coming from another language and want to get up and running with JavaScript as a language for scripting the browser, take the interactive jQuery Air courses on CodeSchool.[7]
- If you want a more formal introduction to the JavaScript language, absorb Marijn Haverbeke's *Eloquent JavaScript*.[8]
- If you're a JavaScript beginner who wants to level up and avoid common pitfalls, spend some time in the JavaScript Garden.[9]

Where to Turn for Help?

When pondering questions like "Should I use `typeof` or `instanceof` here?" steer clear of the dated W3Schools site (which, regrettably, tends to be favored by Google searches). Instead, head to the Mozilla Developer Network (MDN).[10]

5. https://github.com/raganwald/homoiconic/blob/master/2012/01/captain-obvious-on-javascript.md
6. http://www.codecademy.com/
7. http://www.codeschool.com/
8. http://eloquentjavascript.net/
9. http://javascriptgarden.info/
10. https://developer.mozilla.org/

The Mozilla Foundation (you may have heard of its browser, Firefox) is headed up by Brendan Eich, the creator of JavaScript. The foundation knows its stuff.

If you can't find your answer among MDN's pages, take your question to Stack Overflow.[11] The site has fostered an amazingly helpful developer community, and it's a safe bet that any coherent question tagged JavaScript will receive a punctual response.

Running the Code Examples

This book is a bit unusual, in that I discuss both client-side (browser) and server-side (Node.js) code. That reflects the uniquely portable nature of JavaScript. The central concepts apply to all JavaScript environments, but certain examples are aimed at one or the other.

Even if you have no interest in writing Node applications, I hope you'll follow along by running these code snippets locally. See *Running Code in Node.js,* on page x for directions.

Which Examples Are Runnable?

When you see a code snippet with a filename, that means it's self-contained and can be run without modification. Here's an example:

Preface/stringConstructor.js
```
console.log('str'.constructor.name);
```

The surrounding context should make it clear whether the code is runnable in the browser, in Node.js, or in both.

When a code snippet doesn't have a filename, that means it's not self-contained. It may be part of a larger example, or it may be a hypothetical. Here's an example:

```
var tenSeconds = 10 * 1e3;
setTimeout(launchSatellite, tenSeconds);
```

These examples are meant to be read, not run.

Running Code in Node.js

Node is very easy to install and use: just head to http://nodejs.org/, click Download, and run the Windows or OS X installer (or build from source on *nix). You can then run node from the command line to open a JavaScript REPL (analogous to Ruby's irb environment).

11. http://stackoverflow.com/

```
$ node
> Math.pow(5, 6)
15625
```

You can run a JavaScript file by giving its name as an argument to the node command.

```
$ echo "console.log(typeof NaN)" > foo.js
$ node foo.js
number
```

Running Code in the Browser

Every modern browser provides a nice little REPL that lets you run JavaScript code in the context of the current page. But for playing with multiline code examples, you're better off using a web sandbox like jsFiddle.[12]

With jsFiddle, you can enter JavaScript, HTML, and CSS, and then click Run (or press Ctrl+Enter) to see the result. (console output will go to your developer console.) You can bring in a framework like jQuery by choosing it in the left sidebar. And you can save your work, giving you a shareable URL.

Code Style in This Book

JavaScript has no official style guide, but maintaining a consistent style within a project is important. For this book, I've adopted the following (very common) conventions:

- Two-space indentation
- camelCase identifiers
- Semicolons at the end of every expression, except function definitions

More esoterically, I've adopted a special convention for indentation in a chain of function calls, based on a proposal by Reg Braithwaite. The rule is, essentially, that two function calls in a chain have the same indentation level if and only if they return the same object. So, for instance, I might write the following:

```
$('#container > ul li.inactive')
.slideUp();
```

jQuery's slideUp method returns the same object that it was called on. Thus, it isn't indented. By contrast:

```
var $paragraphClone = $('p:last')
  .clone();
```

12. http://jsfiddle.net/

Here, the clone method is indented because it returns a different object.

The advantage of this convention is that it clarifies what each function in a chain is returning. Here's a more complex example:

```
$('h1')
  .first()
  .addClass('first')
.end()
  .last()
  .addClass('last');
```

jQuery's first and last filter a set down to its first and last elements, while end undoes the last filter. So, end is unindented because it returns the same value as $('h1'). (last is allowed to occupy the same indentation level as first because the chain was reset.)

This approach to indentation is especially useful when we're doing functional programming, as we'll see in Chapter 4, *Flow Control with Async.js*, on page ?.

```
[1, 2, 3, 4, 5]
  .filter(function(int) { return int % 2 === 1; })
    .forEach(function(odd) { console.log(odd); })
```

A Word on altJS

A number of languages compile to JavaScript, making code easier to write. (You can find a fairly comprehensive list at http://altjs.org.) This book isn't about them. It's about writing the best JavaScript code we can without the use of a precompiler. I have nothing against altJS (see the next section), but I believe it's important to understand the underlying language.

Some altJS languages are aimed specifically at "taming" async callbacks by allowing them to be written in a more synchronous style. I've included an overview of these languages in Appendix 1, *Tools for Taming JavaScript*, on page ?.

CoffeeScript

It's no secret that I ♥ CoffeeScript, a beautiful and expressive language that compiles to JavaScript. I use it extensively in my day-to-day work at HubSpot. I've given talks on it at conferences like Railsconf and Øredev. And it was the subject of my first book, *CoffeeScript: Accelerated JavaScript Development.*[13]

13. http://pragprog.com/book/tbcoffee/coffeescript

But when I started writing the book you're reading now, I decided that doing it in CoffeeScript would needlessly limit its appeal. By and large, CoffeeScripters understand JavaScript perfectly well, whereas code like `square = (x) => x * x` might as well be hieroglyphics to JavaScript purists.[14]

So, if you're a CoffeeScripter, my apologies for the curly braces. Rest assured that the lessons you draw from this book will carry over to any altJS language.

Resources for This Book

This book has a website at http://pragprog.com/book/tbajs/async-javascript. There you can download the example code used in the book, get up-to-date information, and ask book-related questions in a friendly forum.

For more general JavaScript-related questions, I (again) heartily recommend Stack Overflow.[15] I have no affiliation with the site, but I am an avid fan with a proud 23,000 reputation points (and counting). Coherent, well-formatted questions there are almost always answered promptly.

Finally, if you want to contact me directly, you can reach me at trevorburnham@gmail.com or on Twitter: @trevorburnham. I'm always happy to hear from my readers.

Enough introduction. Let's get our async on!

14. However, maybe not for long: http://wiki.ecmascript.org/doku.php?id=harmony:arrow_function_syntax.
15. http://stackoverflow.com/