Secrets of the JavaScript Ninja

SECOND EDITION

John Resig
Bear Bibeault
Josip Maras

SAMPLE CHAPTER

MANNING

Secrets of the JavaScript Ninja Second Edition

by John Resig
Bear Bibeault
and Josip Maras

Chapter 1

brief contents

Part 1

Warming up

This part of the book will set the stage for your JavaScript ninja training. In chapter 1, we'll look at the current state of JavaScript and explore some of the environments in which JavaScript code can be executed. We'll put a special focus on the environment where it all began—*the browser*—and we'll discuss some of the best practices when developing JavaScript applications.

Because our exploration of JavaScript will be done in the context of browsers, in chapter 2 we'll teach you about the lifecycle of client-side web applications and how executing JavaScript code fits into this lifecycle.

When you're finished with this part of the book, you'll be ready to embark on your training as a JavaScript ninja!

1 JavaScript is everywhere

This chapter covers

- The core language features of JavaScript
- The core items of a JavaScript engine
- Three best practices in JavaScript development

Let's talk about Bob. After spending a few years learning how to create desktop applications in C++, he graduated as a software developer in the early 2000s and then went out into the wide world. At that point, the web had just hit its stride, and everybody wanted to be the next Amazon. So the first thing he did was learn web development.

He learned some PHP so that he could dynamically generate web pages, which he usually sprinkled with JavaScript in order to achieve complex functionality such as form validation and even dynamic on-page clocks! Fast-forward a couple of years, and smartphones had become a thing, so anticipating a large new market opening up, Bob went ahead and learned Objective-C and Java to develop mobile apps that run on iOS and Android.

Over the years, Bob has created many successful applications that all have to be maintained and extended. Unfortunately, jumping daily between all these different programming languages and application frameworks has really started to wear down poor Bob.

Now let's talk about Ann.Two years ago, Ann graduated with a degree in software development, specializing in web- and cloud-based applications. She has created a few medium-sized web applications based on modern Model–view–controller (MVC) frameworks, along with accompanying mobile applications that run on iOS and Android. She has created a desktop application that runs on Linux, Windows, and OS X, and has even started building a serverless version of that application entirely based in the cloud. And *everything she has done has been written in JavaScript*.

That's extraordinary! What took Bob 10 years and 5 languages to do, Ann has achieved in 2 years and in *just one language*. Throughout the history of computing, it has been rare for a particular knowledge set to be so easily transferable and useful across so many different domains.

What started as a humble 10-day project back in 1995 is now one of the most widely used programming languages in the world. JavaScript is quite literally *everywhere*, thanks to more-powerful JavaScript engines and the introduction of frameworks such as Node, Apache Cordova, Ionic, and Electron, which have taken the language beyond the humble web page. And, like HTML, the language itself is now getting long overdue upgrades intended to make JavaScript even more suitable for modern application development.

In this book, we're going to make sure you know all you need to know about JavaScript so that, whether you're like Ann or like Bob, you can develop all sorts of applications on a green field or a brown field.

Do you know?

What are Babel and Traceur, and why are they important to today's JavaScript developers?

What are the core parts of any web browser's JavaScript API used by web applications?

1.1 Understanding the JavaScript language

As they advance through their careers, many JavaScript coders like Bob and Ann reach the point where they're actively using the vast number of elements that form the language. In many cases, however, those skills may not be taken beyond fundamental levels. Our guess is that this is often because JavaScript, using a C-like syntax, bears a surface resemblance to other widespread C-like languages (such as C# and Java), and thus leaves the impression of familiarity.

People often feel that if they know C# or Java, they already have a pretty solid understanding of how JavaScript works. But it's a trap! When compared to other mainstream languages, JavaScript is much more *functionally* oriented. Some JavaScript concepts differ fundamentally from those of most other languages.

These differences include the following:

- *Functions are first-class objects*—In JavaScript, functions coexist with, and can be treated like, any other JavaScript object. They can be created through literals, referenced by variables, passed around as function arguments, and even returned as function return values. We devote much of chapter 3 to exploring some of the wonderful benefits that functions as first-class objects bring to our JavaScript code.
- *Function closures*—The concept of function closures is generally poorly understood, but at the same time it fundamentally and irrevocably exemplifies the importance of functions to JavaScript. For now, it's enough to know that a function is *a closure when it actively maintains ("closes over") the external variables used in its body*. Don't worry for now if you don't see the many benefits of closures; we'll make sure all is crystal clear in chapter 5. In addition to closures, we'll thoroughly explore the many aspects of functions themselves in chapters 3 and 4, as well as identifier scopes in chapter 5.
- *Scopes*—Until recently, JavaScript didn't have block-level variables (as in other C-like languages); instead, we had to rely only on global variables and function-level variables.
- *Prototype-based object orientation*—Unlike other mainstream programming languages (such as C#, Java, and Ruby), which use class-based object orientation, JavaScript uses prototypes. Often, when developers come to JavaScript from class-based languages (such as Java), they try to use JavaScript as if it were Java, essentially writing Java's class-based code using the syntax of JavaScript. Then, for some reason, they're surprised when the results differ from what they expect. This is why we'll go deep into prototypes, how prototype-based object-orientation works, and how it's implemented in JavaScript.

JavaScript consists of a close relationship between objects and prototypes, and functions and closures. Understanding the strong relationships between these concepts can vastly improve your JavaScript programming ability, giving you a strong foundation for any type of application development, regardless of whether your JavaScript code will be executed in a web page, in a desktop app, in a mobile app, or on the server.

In addition to these fundamental concepts, other JavaScript features can help you write more elegant and more efficient code. Some of these are features that seasoned developers like Bob will recognize from other languages, such as Java and C++. In particular, we focus on the following:

- *Generators*, which are functions that can generate multiple values on a per-request basis and can suspend their execution between requests
- *Promises*, which give us better control over asynchronous code
- *Proxies,* which allow us to control access to certain objects
- *Advanced array methods,* which make array-handling code much more elegant

- *Maps*, which we can use to create dictionary collections; and *sets*, which allow us to deal with collections of unique items
- *Regular expressions*, which let us simplify what would otherwise be complicated pieces of code
- *Modules*, which we can use to break code into smaller, relatively self-contained pieces that make projects more manageable

Having a deep understanding of the fundamentals and learning how to use advanced language features to their best advantage can elevate your code to higher levels. Honing your skills to tie these concepts and features together will give you a level of understanding that puts the creation of any type of JavaScript application within your reach.

1.1.1 How will JavaScript evolve?

The ECMAScript committee, in charge of standardizing the language, has just finished the ES7/ES2016 version of JavaScript. The ES7 version is a relatively small upgrade to JavaScript (at least, when compared to ES6), because the committee's goal going forward is to focus on smaller, yearly incremental changes to the language.

In this book we thoroughly explore ES6 but also focus on ES7 features, such as the new async function, which will help you deal with asynchronous code (discussed in chapter 6).

ES6 ES7

NOTE When we cover features of JavaScript defined in ES6/ES2015 or ES7/ES2016, you'll see these icons alongside a link to information about whether they're supported by your browser.

Yearly incremental updates to the language specification are excellent news, but this doesn't mean that web developers will instantly have access to the new features after the specification has been released. JavaScript code has to be executed by a JavaScript engine, so we're often left waiting impatiently for updates to our favorite JavaScript engines that incorporate these new and exciting features.

Unfortunately, although the JavaScript engine developers are trying to keep up and are doing better all the time, there's always a chance that you'll run into features that you are dying to use but that are yet to be supported.

Luckily, you can keep up with the state of feature support in the various browsers via the lists at https://kangax.github.io/compat-table/es6/, http://kangax.github.io/compat-table/es2016plus/, and https://kangax.github.io/compat-table/esnext/.

1.1.2 Transpilers give us access to tomorrow's JavaScript today

Because of the rapid release cycles of browsers, we usually don't have to wait long for a JavaScript feature to be supported. But what happens if we want to take advantage of all the benefits of the newest JavaScript features but are taken hostage by a harsh reality: The users of our web applications may still be using older browsers?

One answer to this problem is to use *transpilers* ("transformation + compiling"), tools that take cutting-edge JavaScript code and transform it into equivalent (or, if that's not possible, similar) code that works properly in most current browsers.

Today's most popular transpilers are Traceur (https://github.com/google/traceur-compiler) and Babel (https://babeljs.io/). Setting them up is easy; just follow one of the tutorials, such as https://github.com/google/traceur-compiler/wiki/Getting-Started or http://babeljs.io/docs/setup/.

In this book, we put a special focus on running JavaScript code in the browser. To effectively use the browser platform, you have to get your hands dirty and study the inner workings of browsers. Let's get started!

1.2 Understanding the browser

These days, JavaScript applications can be executed in many environments. But the environment from which it all began, the environment from which all other environments have taken ideas, and the environment on which we'll focus, is the *browser*. The browser provides various concepts and APIs to thoroughly explore; see figure 1.1.

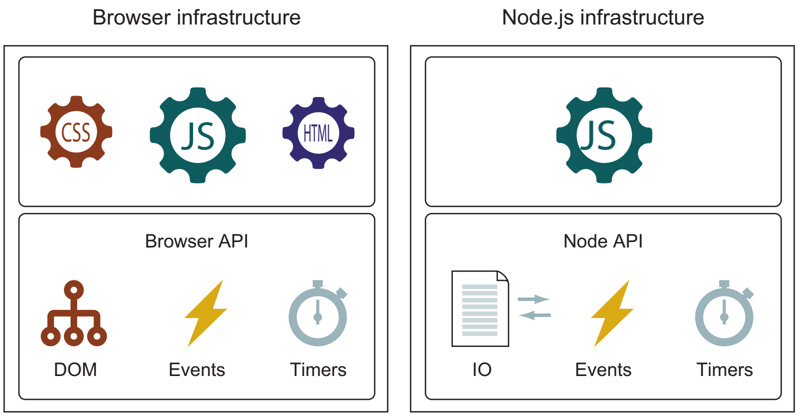

Figure 1.1 Client-side web applications rely on the infrastructure provided by the browser. We'll particularly focus on the DOM, events, timers, and browser APIs.

We'll concentrate on the following:

- *The Document Object Model* (DOM)—The DOM is a structured representation of the UI of a client-side web application that is, at least initially, built from the HTML code of a web application. To develop great applications, you need to not only have a deep understanding of the core JavaScript mechanics, but also study how the DOM is constructed (chapter 2) and how to write effective code that manipulates the DOM (chapter 12). This will put the creation of advanced, highly dynamic UIs at your fingertips.

- *Events*—A huge majority of JavaScript applications are *event-driven* applications, meaning that most of the code is executed in the context of a response to a particular event. Examples of events include network events, timers, and user-generated events such as clicks, mouse moves, keyboard presses, and so on. For this reason, we'll thoroughly explore the mechanisms behind events in chapter 13. We'll pay special attention to *timers*, which are frequently a mystery but let us tackle complex coding tasks such as long-running computations and smooth animations.
- *Browser API*—To help us interact with the world, the browser offers an API that allows us to access information about devices, store data locally, or communicate with remote servers. We'll explore some of these APIs throughout the book.

Perfecting your JavaScript programming skills and achieving deep understanding of APIs offered by the browser will take you far. But sooner, rather than later, you'll run face first into *the browsers* and their various issues and inconsistencies. In a perfect world, all browsers would be bug-free and would support web standards in a consistent fashion; unfortunately, we don't live in that world.

The quality of browsers has improved greatly as of late, but they all still have some bugs, missing APIs, and browser-specific quirks that we need to deal with. Developing a comprehensive strategy for tackling these browser issues, and becoming intimately familiar with their differences and quirks, can be almost as important as proficiency in JavaScript itself.

When we're writing browser applications or JavaScript libraries to be used in them, choosing which browsers to support is an important consideration. We'd like to support them all, but limitations on development and testing resources dictate otherwise. For this reason, we'll thoroughly explore strategies for cross-browser development in chapter 14.

Developing effective, cross-browser code can depend significantly on the skill and experience of the developers. This book is intended to boost that skill level, so let's get to it by looking at current best practices.

1.3 Using current best practices

Mastery of the JavaScript language and a grasp of cross-browser coding issues are important parts of becoming an expert web application developer, but they're not the complete picture. To enter the big leagues, you also need to exhibit the traits that scores of previous developers have proven are beneficial to the development of quality code. These traits are known as *best practices*, and in addition to mastery of the language, they include such elements as

- Debugging skills
- Testing
- Performance analysis

It's vitally important to adhere to these practices when coding, and we'll use them throughout the book. Let's examine some of them next.

1.3.1 Debugging

Debugging JavaScript used to mean using `alert` to verify the value of variables. Fortunately, the ability to debug JavaScript code has dramatically improved, in no small part because of the popularity of the Firebug developer extension for Firefox. Similar tools have been developed for all major browsers:

- *Firebug*—The popular developer extension for Firefox that got the ball rolling (http://getfirebug.com/)
- *Chrome DevTools*—Developed by the Chrome team and used in Chrome and Opera
- *Firefox Developer Tools*—A tool included in Firefox, built by the Firefox team
- *F12 Developer Tools*—Included in Internet Explorer and Microsoft Edge
- *WebKit Inspector*—Used by Safari

As you can see, every major browser offers developer tools that we can use to debug our web applications. The days of using JavaScript alerts for debugging are long gone!

All of these tools are based on similar ideas, which were mostly introduced by Firebug, so they offer similar functionality: exploring the DOM, debugging JavaScript, editing CSS styles, tracking network events, and so on. Any of them will do a fine job; use the one offered by your browser of choice, or in the browser in which you're investigating bugs.

In addition, you can use some of them, such as Chrome Dev Tools, to debug other kinds of applications, like Node.js apps. (We'll introduce you to some debugging techniques in appendix B.)

1.3.2 Testing

Throughout this book, we'll apply testing techniques to ensure that the example code operates as intended and to serve as examples of how to test code in general. The primary tool we'll use for testing is an `assert` function, whose purpose is to assert that a premise is either true or false. By specifying assertions, we can check whether the code is behaving as expected.

The general form of this function is as follows:

```
assert(condition, message);
```

The first parameter is a condition that should be true, and the second is a message that will be displayed if it's not.

Consider this, for example:

```
assert(a === 1, "Disaster! a is not 1!");
```

If the value of variable `a` isn't equal to `1`, the assertion fails, and the somewhat overly dramatic message is displayed.

> **NOTE** The `assert` function isn't a standard feature of the language, so we'll implement it ourselves in appendix B.

1.3.3 Performance analysis

Another important practice is performance analysis. The JavaScript engines have made astounding strides in the performance of JavaScript, but that's no excuse for writing sloppy and inefficient code.

We'll use code such as the following later in this book to collect performance information:

```
console.time("My operation");               <-- Starts the timer

for(var n = 0; n < maxCount; n++){          | Performs
  /*perform the operation to be measured*/  | the operation
}                                           | multiple times

console.timeEnd("My operation");            <-- Stops the timer
```

Here, we bracket the execution of the code to be measured with two calls to the `time` and `timeEnd` methods of the built-in `console` object.

Before the operation begins executing, the call to `console.time` starts a timer with a name (in this case, `My operation`). Then we run the code in the `for` loop a certain number of times (in this case, `maxCount` times). Because a single operation of the code happens much too quickly to measure reliably, we need to perform the code many times to get a measurable value. Frequently, this count can be in the tens of thousands, or even millions, depending on the nature of the code being measured. A little trial and error lets us choose a reasonable value.

When the operation ends, we call the `console.timeEnd` method with the same name. This causes the browser to output the time that elapsed since the timer was started.

These best-practice techniques, along with others you'll learn along the way, will greatly enhance your JavaScript development. Developing applications with the restricted resources that a browser provides, coupled with the increasingly complex world of browser capability and compatibility, requires a robust and complete set of skills.

1.4 Boosting skill transferability

When Bob was first learning web development, each browser had its own way of interpreting script and UI styles, preaching that their way was the best way and making every developer grind their teeth in frustration. Fortunately, the browser wars ended with HTML, CSS, the DOM API, and JavaScript all being standardized, and developer focus turning toward effective cross-browser JavaScript applications. Indeed, this focus on treating websites as applications led to many ideas, tools, and techniques crossing over from desktop applications to web applications. And now, that knowledge and tools transfer has happened again as ideas, tools, and techniques that originated in client-side web development have also permeated other application domains.

Achieving a deep understanding of fundamental JavaScript principles with the knowledge of core APIs can therefore make you a more versatile developer. By using the browsers and Node.js (an environment derived from the browser), you can develop almost any type of application imaginable:

- *Desktop applications*, by using, for example, NW.js (http://nwjs.io/) or Electron (http://electron.atom.io/). These technologies usually wrap the browser so that we can build desktop UIs with standard HTML, CSS, and JavaScript (that way, we can rely on our core JavaScript and browser knowledge), with additional support that makes it possible to interact with the filesystem. We can build truly platform-independent desktop applications that have the same look and feel on Windows, Mac, and Linux.
- *Mobile apps with frameworks*, such as Apache Cordova (https://cordova.apache.org/). Similar to desktop apps built with web technologies, frameworks for mobile apps use a wrapped browser but with additional platform-specific APIs that let us interact with the mobile platform.
- *Server-side applications and applications for embedded devices with Node.js*, an environment derived from the browser that uses many of the same underlying principles as the browser. For example, Node.js executes JavaScript code and relies on events.

Ann doesn't know how lucky she is (although Bob has a pretty good idea). It doesn't matter whether she needs to build a standard desktop application, a mobile application, a server-side application, or even an embedded application—all these types of applications share some of the same underlying principles of standard client-side web applications. By understanding how the core mechanics of JavaScript work, and by understanding the core APIs provided by browsers (such as events, which also have a lot in common with mechanisms provided by Node.js), she can boost her development skills across the board. As can you. In the process, you'll become a more versatile developer and gain the knowledge and understanding to tackle a wide variety of problems. You'll even be able to build your own serverless applications based in the cloud by using JavaScript APIs for services such as AWS Lambda to deploy, maintain, and control your application's cloud components.

1.5 Summary

- Client-side web applications are among the most popular today, and the concepts, tools, and techniques once used only for their development have permeated other application domains. Understanding the foundations of client-side web applications will help you develop applications for a wide variety of domains.

- To improve your development skills, you have to gain a deep understanding of the core mechanics of JavaScript, as well as the infrastructure provided by the browsers.
- This book focuses on core JavaScript mechanisms such as functions, function closures, and prototypes, as well as on new JavaScript features such as generators, promises, proxies, maps, sets, and modules.
- JavaScript can be executed in a large number of environments, but the environment where it all began, and the environment we'll concentrate on, is the browser.
- In addition to JavaScript, we'll explore browser internals such as the DOM (a structured representation of the web page UI) and events, because client-side web applications are event-driven applications.
- We'll do this exploration with best practices in mind: debugging, testing, and performance analysis.

JAVASCRIPT/WEB DEVELOPMENT

Secrets of the JavaScript Ninja Second Edition

Resig • Bibeault • Maras

Free eBook
SEE INSERT

JavaScript is rapidly becoming a universal language for every type of application, whether on the web, on the desktop, in the cloud, or on mobile devices. When you become a JavaScript pro, you have a powerful skill set that's usable across all these domains.

Secrets of the JavaScript Ninja, Second Edition uses practical examples to clearly illustrate each core concept and technique. This completely revised edition shows you how to master key JavaScript concepts such as functions, closures, objects, prototypes, and promises. It covers APIs such as the DOM, events, and timers. You'll discover best practice techniques such as testing, and cross-browser development, all taught from the perspective of skilled JavaScript practitioners.

What's Inside

- Writing more effective code with functions, objects, and closures
- Learning to avoid JavaScript application pitfalls
- Using regular expressions to write succinct text-processing code
- Managing asynchronous code with promises
- Fully revised to cover concepts from ES6 and ES7

You don't have to be a ninja to read this book—just be willing to become one. Are you ready?

John Resig is an acknowledged JavaScript authority and creator of the jQuery library. Bear Bibeault is a web developer and author of the first edition, as well as coauthor of *Ajax in Practice*, *Prototype and Scriptaculous in Action*, and *jQuery in Action*. Josip Maras is a post-doctoral researcher and teacher.

"Essential reading for developers of any discipline ... with powerful techniques to improve your JavaScript."
—Becky Huett, Big Shovel Labs

"Excellent and comprehensive insight into the magic of functions and closures for the efficient use of JavaScript."
—Gerd Klevesaat, Siemens

"The essential resource for moving your JavaScript skills to the next level."
—David Starkey, Blum

"Helps you master both the stealthy and bold techniques of modern JavaScript."
—Christopher Haupt
New Relic Inc.

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit
manning.com/books/secrets-of-the-javascript-ninja-second-edition

ISBN-13: 978-1-61729-285-9
ISBN-10: 1-61729-285-0
54499
9 781617 292859

MANNING $44.99 / Can $51.99 [INCLUDING eBOOK]