SAMPLE CHAPTER

Secrets of the JavaScript Ninja

John Resig
Bear Bibeault

MANNING

Secrets of the JavaScript Ninja

by John Resig
Bear Bibeault

Chapter 1

brief contents

1 Enter the ninja

This chapter covers

- A look at the purpose and structure of this book
- Which libraries we'll look at
- What is "advanced" JavaScript programming?
- Cross-browser authoring
- Test suite examples

If you're reading this book, you know that there's nothing simple about creating effective and cross-browser JavaScript code. In addition to the normal challenges of writing clean code, we have the added complexity of dealing with obtuse browser differences and complexities. To deal with these challenges, JavaScript developers frequently capture sets of common and reusable functionality in the form of JavaScript libraries.

These libraries vary widely in approach, content, and complexity, but one constant remains: they need to be easy to use, incur the least amount of overhead, and be able to work across all browsers that we wish to target.

It stands to reason, then, that understanding how the very best JavaScript libraries are constructed can provide us with great insight into how our own code

can be constructed to achieve these same goals. This book sets out to uncover the techniques and secrets used by these world-class code bases and to gather them into a single resource.

In this book we'll be examining the techniques that were (and continue to be) used to create the popular JavaScript libraries. Let's meet those libraries!

1.1 The JavaScript libraries we'll be tapping

The techniques and practices used to create modern JavaScript libraries will be the focus of our attention in this book. The primary library that we'll be considering is, of course, jQuery, which has risen in prominence to be the most ubiquitous JavaScript library in modern use.

jQuery (http://jquery.com) was created by John Resig and released in January of 2006. jQuery popularized the use of CSS selectors to match DOM content. Among its many capabilities, it provides DOM manipulation, Ajax, event handling, and animation functionality.

This library has come to dominate the JavaScript library market, being used on hundreds of thousands of websites, and interacted with by millions of users. Through considerable use and feedback, this library has been refined over the years—and continues to evolve—into the optimal code base that it is today.

In addition to examining example code from jQuery, we'll also look at techniques utilized by the following libraries:

- Prototype (http://prototypejs.org/)—The godfather of the modern JavaScript libraries, created by Sam Stephenson and released in 2005. This library embodies DOM, Ajax, and event functionality, in addition to object-oriented, aspect-oriented, and functional programming techniques.
- Yahoo! UI (http://developer.yahoo.com/yui)—The result of internal JavaScript framework development at Yahoo! and released to the public in February of 2006. Yahoo! UI (YUI) includes DOM, Ajax, event, and animation capabilities in addition to a number of preconstructed widgets (calendar, grid, accordion, and others).
- base2 (http://code.google.com/p/base2)—Created by Dean Edwards and released in March 2007. This library supports DOM and event functionality. Its claim to fame is that it attempts to implement the various W3C specifications in a universal, cross-browser manner.

All of these libraries are well constructed and tackle their target problem areas comprehensively. For these reasons, they'll serve as a good basis for further analysis, and understanding the fundamental construction of these code bases will give us insight into the process of world-class JavaScript library construction.

But these techniques aren't only useful for constructing large libraries; they can be applied to all JavaScript coding, regardless of size.

The makeup of a JavaScript library can be broken down into three aspects:

- Advanced use of the JavaScript language
- Meticulous construction of cross-browser code
- The use of current best practices that tie everything together

We'll be carefully analyzing these three aspects in each of the libraries to gather a complete knowledge base we can use to create our own effective JavaScript code.

1.2 Understanding the JavaScript language

Many JavaScript coders, as they advance through their careers, may get to the point where they're actively using the vast array of elements comprising the language, including objects and functions and (if they've been paying attention to coding trends) even anonymous inline functions. In many cases, however, those skills may not be taken beyond fundamental levels. Additionally, there's generally a very poor understanding of the purpose and implementation of *closures* in JavaScript, which fundamentally and irrevocably exemplify the importance of functions to the language.

JavaScript consists of a close relationship between objects, functions, and closures (see figure 1.1). Understanding the strong relationship between these three concepts can vastly improve our JavaScript programming ability, giving us a strong foundation for any type of application development.

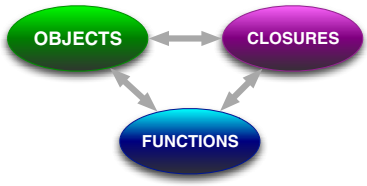

Figure 1.1 JavaScript consists of a close relationship between objects, functions, and closures.

Many JavaScript developers, especially those coming from an object-oriented background, may pay a lot of attention to objects, but at the expense of understanding how functions and closures contribute to the big picture.

In addition to these fundamental concepts, there are two features in JavaScript that are woefully underused: timers and regular expressions. These two concepts have applications in virtually any JavaScript code base, but they aren't always used to their full potential due to their misunderstood nature.

A firm grasp of how timers operate within the browser, all too frequently a mystery, gives us the ability to tackle complex coding tasks such as long-running computations and smooth animations. And a sound understanding of how regular expressions work allows us to simplify what would otherwise be quite complicated pieces of code.

As another high point of our advanced tour of the JavaScript language, we'll take a look at the `with` statement in chapter 10, and the divisive `eval()` method in chapter 9—two important, but controversial, language features that have been trivialized, misused, and even condemned outright by many JavaScript programmers.

NOTE Those of you who have been keeping track of what's moving and shaking in the web development world will know that both of these topics are controversial and are either deprecated or limited in future versions of JavaScript.

But as you'll likely come across these concepts in existing code, it's important to understand them, even if you have no plans to use them in future code.

By looking at the work of some of the best JavaScript coders, we'll see that, when used appropriately, advanced language features allow for the creation of some fantastic pieces of code that wouldn't be otherwise possible. To a large degree, these advanced features can also be used for some interesting metaprogramming exercises, molding JavaScript into whatever we want it to be.

Learning how to use advanced language features responsibly and to their best advantage can certainly elevate our code to higher levels, and honing our skills to tie these concepts and features together will give us a level of understanding that puts the creation of any type of JavaScript application within our reach. This foundation will give us a solid base for moving forward, starting with writing solid, cross-browser code.

1.3 Cross-browser considerations

Perfecting our JavaScript programming skills will take us far, especially now that JavaScript has escaped the confines of the browser and is being used on the server with JavaScript engines like Rhino and V8 and libraries like Node.js. But when developing browser-based JavaScript applications (which is the focus of this book), sooner rather than later, we're going to run face first into *The Browsers* and their maddening issues and inconsistencies.

In a perfect world, all browsers would be bug-free and would support web standards in a consistent fashion, but we all know that we most certainly 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'll need to deal with. Developing a comprehensive strategy for tackling these browser issues, and becoming intimately familiar with their differences and quirks, is just as important, if not more so, than proficiency in JavaScript itself.

When writing browser applications or JavaScript libraries to be used in them, picking and choosing which browsers to support is an important consideration. We'd probably like to support them all, but limitations on development and testing resources dictate otherwise. So how do we decide which to support, and to what level?

An approach that we can employ is one loosely borrowed from an older Yahoo! approach that was called *graded browser support*. In this technique, we create a browser support matrix that serves as a snapshot of how important a browser and its platform are to our needs.

In such a table, we list the target platforms on one axis, and the browsers on the other. Then, in the table cells, we give a "grade" (A through F, or any other grading system that meets our needs) to each browser/platform combination. Table 1.1 shows a hypothetical example of such a table.

Note that we haven't filled in any grades. What grades you assign to a particular combination of platform and browser is entirely dependent upon the needs and requirements of your project, as well as other important factors, like the makeup of

Table 1.1 A hypothetical "browser support matrix"

	Windows	OS X	Linux	iOS	Android
IE 6		N/A	N/A	N/A	N/A
IE 7, 8		N/A	N/A	N/A	N/A
IE 9		N/A	N/A	N/A	N/A
Firefox				N/A	
Chrome					
Safari			N/A		N/A
Opera					

the target audience. We can use this approach to come up with grades that measure how important support for the platform/browser is, and combine that info with the cost of that support to try to come up with the optimal set of supported browsers. We'll be exploring this in more depth in chapter 11.

As it's impractical to develop against a large number of platform/browser combinations, we must weigh the costs versus the benefits of supporting the various browsers. Any such analysis must take into account multiple considerations, the primary of which are

- The expectations and needs of the target audience
- The market share of the browser
- The amount of effort necessary to support the browser

The first point is a subjective one that only your project can determine. Market share, on the other hand, can frequently be measured using available information. And a rough estimate of the effort involved in supporting each browser can be determined by considering the capabilities of the browsers and their adherence to modern standards.

Figure 1.2 shows a sample chart that represents information on browser usage (obtained from StatCounter for August 2012) and our personal opinions on the cost of development for the top desktop browsers.

Charting the benefit versus cost in this manner shows at a glance where we can put our effort to get the most "bang for the buck." Here are a few things that jump out of this chart:

- Even though it's relatively a lot more effort to support Internet Explorer 7 and 8 than the standards-compliant browsers, they still have a large market share, which makes the extra effort worthwhile if those users are an important target for our application audience.
- IE 9, having made great strides towards standards compliance, is easier to support than previous versions of IE, and it's already making headway into market share.

- Supporting Firefox and Chrome is a no-brainer, because they have a large market share and are easy to support.
- Even though Safari has a relatively low market share, it still deserves support, as its standards-compliant nature makes its cost small. (As a rule of thumb, if it works in Chrome, it'll likely work in Safari—pathological cases notwithstanding.)
- Opera, though it requires no more effort than Safari, can lose out on the desktop because of its minuscule market share. But if the mobile platforms are important to you, mobile Opera is a bigger player; see figure 1.3.
- Nothing really need be said about IE 6. (See www.ie6countdown.com.)

Things change pretty drastically when we take a look at the mobile landscape, as shown in figure 1.3.

Of course, nothing is ever quite so cut-and-dried. It might be safe to say that benefit is more important than cost, but it ultimately comes down to the choices of those in the decision-making process, taking into account factors such as the needs of the market and other business concerns. But quantifying the costs versus benefits is a good starting point for making these important support decisions.

Also, be aware that the landscape changes rapidly. Keeping tabs on sites such as http://gs.statcounter.com is a wise precaution.

Another possible factor for resource-constrained organizations is the skill of the development team. While the primary reason for building an app is its use by end users, developers may have to build the skills necessary to develop the application to meet the end users' needs. Such considerations need to be taken into account during the cost analysis phase.

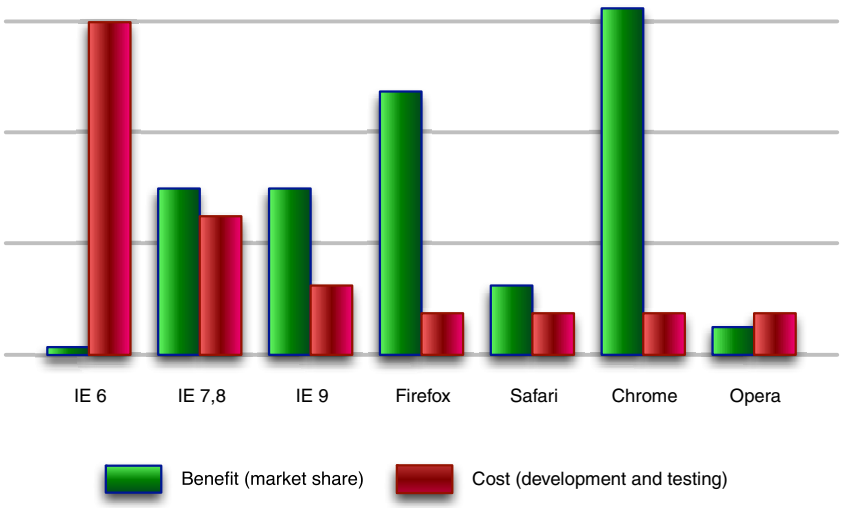

Figure 1.2 Analyzing the cost versus the benefit of supporting various desktop browsers indicates where we should put our effort.

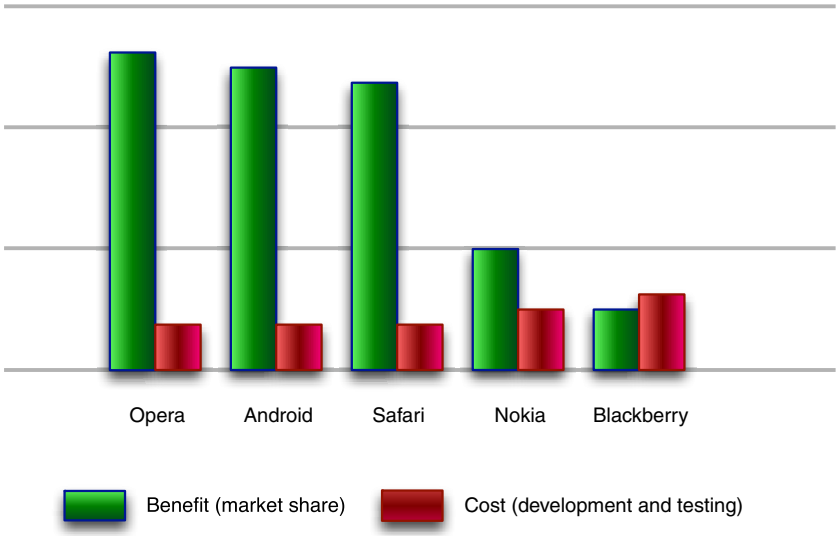

Figure 1.3 The mobile landscape, where development costs are fairly even, comes down to usage statistics.

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

1.4 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 proved are beneficial to the development of quality code. These traits, which we'll examine in depth in chapter 2, are known as *best practices* and, in addition to mastery of the language, include such elements as

- Testing
- Performance analysis
- Debugging skills

It's vitally important to adhere to these practices in our coding, *and* frequently; the complexity of cross-browser development certainly justifies it. Let's examine a couple of these practices.

1.4.1 Current best practice: testing

Throughout this book, we'll be applying a number of testing techniques that serve to ensure that our example code operates as intended, as well as to serve as examples of

how to test general code. The primary tool that we'll be using for testing is an `assert()` function, whose purpose is to assert that a premise is either true or false.

The general form of this function is

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

where 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 that the `assert()` function isn't an innate feature of the language (some languages, such as Java, provide such capabilities), so we'll be implementing it ourselves. We'll be discussing its implementation and use in chapter 2.

1.4.2 Current best practice: performance analysis

Another important practice is performance analysis. The JavaScript engines in the browsers have been making astounding strides in the performance of JavaScript itself, but that's no excuse for us to write sloppy and inefficient code.

We'll be using code such as the following later in this book for collecting performance information:

```
start = new Date().getTime();
for (var n = 0; n < maxCount; n++) {
  /* perform the operation to be measured *//
}
elapsed = new Date().getTime() - start;
assert(true,"Measured time: " + elapsed);
```

Here, we bracket the execution of the code to be measured with the collection of timestamps: one before we execute the code and one after. Their difference tells us how long the code took to perform, which we can compare against alternatives to the code that we measure using the same technique.

Note how we perform the code multiple times; in this example, we perform it the number of times represented by `maxCount`. 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 upon the nature of the code being measured. A little trial-and-error lets us choose a reasonable value.

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

1.5 Summary

Here's a rundown of what we've learned in this chapter:

- Cross-browser web application development is hard, harder than most people would think.
- In order to pull it off, we need not only a mastery of the JavaScript language, but a thorough knowledge of the browsers, along with their quirks and inconsistencies, and a good grounding in standard current best practices.
- While JavaScript development can certainly be challenging, there are those brave souls who have already gone down this tortuous route: the developers of JavaScript libraries. We'll be distilling the knowledge demonstrated in the construction of these code bases, effectively fueling our development skills and raising them to world-class level.

This exploration will certainly be informative and educational—let's enjoy the ride!

JAVASCRIPT/WEB DEVELOPMENT

Secrets of the JavaScript Ninja

John Resig • Bear Bibeault

Free eBook
see insert

You can't always attack software head-on. Sometimes you come at it sideways or sneak up from behind. You need to master an arsenal of tools and know every stealthy trick. You have to be a ninja.

Secrets of the JavaScript Ninja leads you down the pathway to JavaScript enlightenment. This unique book starts with key concepts, like the relationships between functions, objects, and closures, taught from the master's perspective. You'll grow from apprentice to ninja as you soak up fresh insights on the techniques you use every day and discover features and capabilities you never knew about. When you reach the final chapters, you'll be ready to code brilliant JavaScript applications and maybe even write your own libraries and frameworks.

What's Inside

- Functions, objects, closures, regular expressions, and more
- Seeing applications and libraries from the right perspective
- Modern JavaScript design
- Dealing with the complexities of cross-browser development

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 the creator of the jQuery library. **Bear Bibeault** is a web developer and coauthor of *Ajax in Practice, Prototype and Scriptaculous in Action*, and *jQuery in Action* from Manning.

"From two masters, the art of crafting effective cross-browser JavaScript."
—Glenn Stokol
Oracle Corporation

"Consistent with the jQuery motto, 'Write less, do more'."
—André Roberge
Université Sainte-Anne

"Interesting and original techniques."
—Scott Sauyet
Four Winds Software

"Read this book and you'll no longer blindly plug in a snippet of code and marvel at how it works—you'll understand *why* it works."
—Joe Litton
Collaborative Software Developer, JoeLitton.net

"Will help you raise your JavaScript to the realm of the masters."
—Christopher Haupt
greenstack.com

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit manning.com/SecretsoftheJavaScriptNinja

ISBN-13: 978-1933988696
ISBN-10: 1933988697
53999
9 781933 988696

MANNING US $39.99/CAN $41.99 [INCLUDING EBOOK]