Solutions & Examples for jQuery Developers

Free Sampler

jQuery Cookbook

O'REILLY®

jQuery Community Experts

O'Reilly Ebooks—Your bookshelf on your devices!

When you buy an ebook through oreilly.com, you get lifetime access to the book, and whenever possible we provide it to you in four, DRM-free file formats—PDF, .epub, Kindle-compatible .mobi, and Android .apk ebook—that you can use on the devices of your choice. Our ebook files are fully searchable and you can cut-and-paste and print them. We also alert you when we've updated the files with corrections and additions.

Learn more at http://oreilly.com/ebooks/

You can also purchase O'Reilly ebooks through iTunes, the Android Marketplace, and Amazon.com.

jQuery Cookbook

jQuery Cookbook

jQuery Community Experts

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Taipei • Tokyo

jQuery Cookbook
by jQuery Community Experts

Printed in the United States of America.

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

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editor: Simon St.Laurent
Production Editor: Sarah Schneider
Copyeditor: Kim Wimpsett
Proofreader: Andrea Fox
Production Services: Molly Sharp

Indexer: Fred Brown
Cover Designer: Karen Montgomery
Interior Designer: David Futato
Illustrator: Robert Romano

Printing History:
November 2009: First Edition.

This book uses RepKover, a durable and flexible lay-flat binding.

ISBN: 978-0-596-15977-1

[S]

1257774409

Table of Contents

Foreword

When I first started work on building jQuery, back in 2005, I had a simple goal in mind: I wanted to be able to write a web application and have it work in all the major browsers—without further tinkering and bug fixing. It was a couple of months before I had a set of utilities that were stable enough to achieve that goal for my personal use. I thought I was relatively done at this point; little did I know that my work was just beginning.

Since those simple beginnings, jQuery has grown and adapted as new users use the library for their projects. This has proven to be the most challenging part of developing a JavaScript library; while it is quite easy to build a library that'll work for yourself or a specific application, it becomes incredibly challenging to develop a library that'll work in as many environments as possible (old browsers, legacy web pages, and strange markup abound). Surprisingly, even as jQuery has adapted to handle more use cases, most of the original API has stayed intact.

One thing I find particularly interesting is to see how developers use jQuery and make it their own. As someone with a background in computer science, I find it quite surprising that so many designers and nonprogrammers find jQuery to be compelling. Seeing how they interact with the library has given me a better appreciation of simple API design. Additionally, seeing many advanced programmers take jQuery and develop large, complex applications with it has been quite illuminating. The best part of all of this, though, is the ability to learn from everyone who uses the library.

A side benefit of using jQuery is its extensible plugin structure. When I first developed jQuery, I was sure to include some simple ways for developers to extend the API that it provided. This has blossomed into a large and varied community of plugins, encompassing a whole ecosystem of applications, developers, and use cases. Much of jQuery's growth has been fueled by this community—without it, the library wouldn't be where it is today, so I'm glad that there are chapters dedicated to some of the most interesting plugins and what you can do with them. One of the best ways to expand your preconceived notion of what you can do with jQuery is to learn and use code from the jQuery plugin community.

This is largely what makes something like a cookbook so interesting: it takes the cool things that developers have done, and have learned, in their day-to-day coding and distills it to bite-sized chunks for later consumption. Personally, I find a cookbook to be one of the best ways to challenge my preconceived notions of a language or library. I love seeing cases where an API that I thought I knew well is turned around and used in new and interesting ways. I hope this book is able to serve you well, teaching you new and interesting ways to use jQuery.

—John Resig
Creator, Lead Developer, jQuery

Contributors

Chapter Authors

Jonathan Sharp has been passionate about the Internet and web development since 1996. Over the years that have followed, he has worked for startups and for Fortune 500 corporations. Jonathan founded Out West Media, LLC, in greater Omaha, Nebraska, and provides frontend engineering and architecture services with a focus on custom XHTML, CSS, and jQuery development. Jonathan is a jQuery core team member and an author and presenter when not coding. Jonathan is most grateful for his wife, Erin; daughter, Noel; two dogs, and two horses.

Rob Burns develops interactive web applications at A Mountain Top, LLC. For the past 12 years he has been exploring website development using a wide range of tools and technologies. In his spare time, he enjoys natural-language processing and the wealth of opportunity in open source software projects.

Rebecca Murphey is an independent frontend architecture consultant, crafting custom frontend solutions that serve as the glue between server and browser. She also provides training in frontend development, with an emphasis on the jQuery library. She lives with her partner, two dogs, and two cats in Durham, North Carolina.

Ariel Flesler is a web developer and a video game programmer. He's been contributing to jQuery since January 2007 and joined the core team in May 2008. He is 23 years old and was born in Buenos Aires, Argentina. He's studying at the National Technological University (Argentina) and is hoping to become a systems analyst by 2010 and a systems engineer by 2012. He started working as an ASP.NET(C#) programmer and then switched to client-side development of XHTML sites and Ajax applications. He's currently working at QB9 where he develops AS3-based casual games and MMOs.

Cody Lindley is a Christian, husband, son, father, brother, outdoor enthusiast, and professional client-side engineer. Since 1997 he has been passionate about HTML, CSS, JavaScript, Flash, interaction design, interface design, and HCI. He is most well known in the jQuery community for the creation of ThickBox, a modal/dialog solution. In 2008 he officially joined the jQuery team as an evangelist. His current focus has been

on client-side optimization techniques as well as speaking and writing about jQuery. His website is *http://www.codylindley.com*.

Remy Sharp is a developer, author, speaker, and blogger. Remy started his professional web development career in 1999 as the sole developer for a finance website and, as such, was exposed to all aspects of running the website during, and long after, the dotcom boom. Today he runs his own development company called Left Logic in Brighton, UK, writing and coding JavaScript, jQuery, HTML 5, CSS, PHP, Perl, and anything else he can get his hands on.

Mike Hostetler is an inventor, entrepreneur, programmer, and proud father. Having worked with web technologies since the mid-1990s, Mike has had extensive experience developing web applications with PHP and JavaScript. Currently, Mike works at the helm of A Mountain Top, LLC, a web technology consulting firm in Denver, Colorado. Heavily involved in open source, Mike is a member of the jQuery core team, leads the QCubed PHP5 Framework project, and participates in the Drupal project. When not in front of a computer, Mike enjoys hiking, fly fishing, snowboarding, and spending time with his family.

Ralph Whitbeck is a graduate of the Rochester Institute of Technology and is currently a senior developer for BrandLogic Corporation in Rochester, New York. His responsibilities at BrandLogic include interface design, usability testing, and web and application development. Ralph is able to program complex web application systems in ASP.NET, C#, and SQL Server and also uses client-side technologies such as XHTML, CSS, and JavaScript/jQuery in order to implement client-approved designs. Ralph officially joined the jQuery team as an evangelist in October 2009. Ralph enjoys spending time with his wife, Hope, and his three boys, Brandon, Jordan, and Ralphie. You can find out more about Ralph on his personal blog.

Nathan Smith is a goofy guy who has been building websites since late last century. He enjoys hand-coding HTML, CSS, and JavaScript. He also dabbles in design and information architecture. He has written for online and paper publications such as Adobe Developer Center, Digital Web, and .NET Magazine. He has spoken at venues including Adobe MAX, BibleTech, Drupal Camp, Echo Conference, Ministry 2.0, Refresh Dallas, and Webmaster Jam Session. Nathan works as a UX developer at FellowshipTech.com. He holds a Master of Divinity degree from Asbury Theological Seminary. He started Godbit.com, a community resource aimed at helping churches and ministries make better use of the Web. He also created the 960 Grid System, a framework for sketching, designing, and coding page layouts.

Brian Cherne is a software developer with more than a decade of experience blueprinting and building web-based applications, kiosks, and high-traffic e-commerce websites. He is also the author of the hoverIntent jQuery plugin. When not geeking out with code, Brian can be found ballroom dancing, practicing martial arts, or studying Russian culture and language.

Jörn Zaefferer is a professional software developer from Cologne, Germany. He creates application programming interfaces (APIs), graphical user interfaces (GUIs), software architectures, and databases, for both web and desktop applications. His work focuses on the Java platform, while his client-side scripting revolves around jQuery. He started contributing to jQuery in mid-2006 and has since cocreated and maintained QUnit, jQuery's unit testing framework; released and maintained a half dozen very popular jQuery plugins; and contributed to jQuery books as both author and tech reviewer. He is also a lead developer for jQuery UI.

James Padolsey is an enthusiastic web developer and blogger based in London, UK. He's been crazy about jQuery since he first discovered it; he's written tutorials teaching it, articles and blog posts discussing it, and plenty of plugins for the community. James' plans for the future include a computer science degree from the University of Kent and a career that allows him to continually push boundaries. His website is *http://james.padolsey.com*.

Scott González is a web application developer living in Raleigh, North Carolina, who enjoys building highly dynamic systems and flexible, scalable frameworks. He has been contributing to jQuery since 2007 and is currently the development lead for jQuery UI, jQuery's official user interface library. Scott also writes tutorials about jQuery and jQuery UI on nemikor.com and speaks about jQuery at conferences.

Michael Geary started developing software when editing code meant punching a paper tape on a Teletype machine, and "standards-compliant" meant following ECMA-10 Standard for Data Interchange on Punched Tape. Today Mike is a web and Android developer with a particular interest in writing fast, clean, and simple code, and he enjoys helping other developers on the jQuery mailing lists. Mike's recent projects include a series of 2008 election result and voter information maps for Google; and StrataLogic, a mashup of traditional classroom wall maps and atlases overlaid on Google Earth. His website is *http://mg.to*.

Maggie Wachs, **Scott Jehl**, **Todd Parker**, and **Patty Toland** are Filament Group. Together, they design and develop highly functional user interfaces for consumer- and business-oriented websites, wireless devices, and installed and web-based applications, with a specific focus on delivering intuitive and usable experiences that are also broadly accessible. They are sponsor and design leads of the jQuery UI team, for whom they designed and developed ThemeRoller.com, and they actively contribute to ongoing development of the official jQuery UI library and CSS Framework.

Richard D. Worth is a web UI developer. He is the release manager for jQuery UI and one of its longest-contributing developers. He is author or coauthor of the Dialog, Progressbar, Selectable, and Slider plugins. Richard also enjoys speaking and consulting on jQuery and jQuery UI around the world. Richard is raising a growing family in Northern Virginia (Washington, D.C. suburbs) with his lovely wife, Nancy. They have been blessed to date with three beautiful children: Naomi, Asher, and Isaiah. Richard's website is *http://rdworth.org/*.

Tech Editors

Karl Swedberg, after having taught high school English, edited copy for an advertising agency, and owned a coffee house, began his career as a web developer four years ago. He now works for Fusionary Media in Grand Rapids, Michigan, where he specializes in client-side scripting and interaction design. Karl is a member of the jQuery project team and coauthor of *Learning jQuery 1.3* and *jQuery Reference Guide* (both published by Packt). You can find some of his tips and tutorials at *http://www.learningjquery.com*.

Dave Methvin is the chief technology officer at PCPitstop.com and one of the founding partners of the company. He has been using jQuery since 2006, is active on the jQuery help groups, and has contributed several popular jQuery plugins including Corner and Splitter. Before joining PC Pitstop, Dave served as executive editor at both *PC Tech Journal* and *Windows Magazine*, where he wrote a column on JavaScript. He continues to write for several PC-related websites including InformationWeek. Dave holds bachelor's and master's degrees in computer science from the University of Virginia.

David Serduke is a frontend programmer who is recently spending much of his time server side. After programming for many years, he started using jQuery in late 2007 and shortly after joined the jQuery core team. David is currently creating websites for financial institutions and bringing the benefits of jQuery to ASP.NET enterprise applications. David lives in northern California where he received a bachelor's degree from the University of California at Berkeley in electrical engineering and an MBA from St. Mary's College.

Scott Mark is an enterprise application architect at Medtronic. He works on web-based personalized information portals and transactional applications with an eye toward maintaining high usability in a regulated environment. His key interest areas at the moment are rich Internet applications and multitouch user interface technologies. Scott lives in Minnesota with his lovely wife, two sons, and a black lab. He blogs about technology at *http://scottmark.wordpress.com* and long-distance trail running at *http://runlikemonkey.com*.

Preface

The jQuery library has taken the frontend development world by storm. Its dead-simple syntax makes once-complicated tasks downright trivial—enjoyable, even. Many a developer has been quickly seduced by its elegance and clarity. If you've started using the library, you're already adding rich, interactive experiences to your projects.

Getting started is easy, but as is the case with many of the tools we use to develop websites, it can take months or even years to fully appreciate the breadth and depth of the jQuery library. The library is chock-full of features you might never have known to wish for. Once you know about them, they can dramatically change how you approach the problems you're called upon to solve.

The goal of this cookbook is to expose you, dear reader, to the patterns and practices of some of the leading frontend developers who use jQuery in their everyday projects. Over the course of 18 chapters, they'll guide you through solutions to problems that range from straightforward to complex. Whether you're a jQuery newcomer or a grizzled JavaScript veteran, you're likely to gain new insight into harnessing the full power of jQuery to create compelling, robust, high-performance user interfaces.

Who This Book Is For

Maybe you're a designer who is intrigued by the interactivity that jQuery can provide. Maybe you're a frontend developer who has worked with jQuery before and wants to see how other people accomplish common tasks. Maybe you're a server-side developer who's frequently called upon to write client-side code.

Truth be told, this cookbook will be valuable to anyone who works with jQuery—or who hopes to work with jQuery. If you're just starting out with the library, you may want to consider pairing this book with *Learning jQuery 1.3* from Packt, or *jQuery in Action* from Manning. If you're already using jQuery in your projects, this book will serve to enhance your knowledge of the library's features, hidden gems, and idiosyncrasies.

What You'll Learn

We'll start out by covering the basics and general best practices—including jQuery in your page, making selections, and traversing and manipulation. Even frequent jQuery users are likely to pick up a tip or two. From there, we move on to real-world use cases, walking you through tried-and-true (and tested) solutions to frequent problems involving events, effects, dimensions, forms, and user interface elements (with and without the help of jQuery UI). At the end, we'll take a look at testing your jQuery applications and integrating jQuery into complex sites.

Along the way, you'll learn strategies for leveraging jQuery to solve problems that go far beyond the basics. We'll explore how to make the most of jQuery's event management system, including custom events and custom event data; how to progressively enhance forms; how to position and reposition elements on the page; how to create user interface elements such as tabs, accordions, and modals from scratch; how to craft your code for readability and maintainability; how to optimize your code to ease testing, eliminate bottlenecks, and ensure peak performance; and more.

Because this is a cookbook and not a manual, you're of course welcome to cherry-pick the recipes you read; the individual recipes alone are worth the price of admission. As a whole, though, the book provides a rare glimpse into the problem-solving approaches of some of the best and brightest in the jQuery community. With that in mind, we encourage you to at least skim it from front to back—you never know which line of code will provide the "Aha!" moment you need to take your skills to the next level.

jQuery Style and Conventions

jQuery places a heavy emphasis on *chaining*—calling methods on element selections in sequence, confident in the knowledge that each method will give you back a selection of elements you can continue to work with. This pattern is explained in depth in Chapter 1—if you're new to the library, you'll want to understand this concept, because it is used heavily in subsequent chapters.

jQuery's features are organized into a handful of simple categories: core functionality, selecting, manipulating, traversing, CSS, attributes, events, effects, Ajax, and utilities. Learning these categories, and how methods fit into them, will greatly enhance your understanding of the material in this book.

One of the best practices this book will cover is the concept of storing element selections in a variable, rather than making the same selection repeatedly. When a selection is stored in a variable, it is commonplace for that variable to begin with the `$` character, indicating that it is a jQuery object. This can make code easier to read and maintain, but it should be understood that starting the variable name with the `$` character is merely a convention; it carries no special meaning, unlike in other languages such as PHP.

In general, the code examples in this book strive for clarity and readability over compactness, so the examples may be more verbose than is strictly necessary. If you see an opportunity for optimization, you should not hesitate to take it. At the same time, you'll do well to strive for clarity and readability in your own code and use minification tools to prepare your code for production use.

Other Options

If you're looking for other jQuery resources, here are some we recommend:

- *Learning jQuery 1.3*, by Jonathan Chaffer, Karl Swedberg, and John Resig (Packt)
- *jQuery in Action*, by Bear Bibeault, Yehuda Katz, and John Resig (Manning)
- *jQuery UI 1.6: The User Interface Library for jQuery*, by Dan Wellman (Packt)

If You Have Problems Making Examples Work

Before you check anything else, ensure that you are loading the jQuery library on the page—you'd be surprised how many times this is the solution to the "It's not working!" problem. If you are using jQuery with another JavaScript library, you may need to use `jQuery.noConflict()` to make it play well with others. If you're loading scripts that require the presence of jQuery, make sure you are loading them after you've loaded the jQuery library.

Much of the code in this book requires the document to be "ready" before JavaScript can interact with it. If you've included code in the head of the document, make sure your code is enclosed in `$(document).ready(function() { ... });` so that it knows to wait until the document is ready for interaction.

Some of the features discussed in this book are available only in jQuery 1.3 and later. If you are upgrading from an older version of jQuery, make sure you've upgraded any plugins you're using as well—outdated plugins can lead to unpredictable behavior.

If you're having difficulty getting an example to work in an existing application, make sure you can get the example working on its own before trying to integrate it with your existing code. If that works, tools such as Firebug for the Firefox browser can be useful in identifying the source of the problem.

If you're including a minified version of jQuery and running into errors that point to the jQuery library itself, you may want to consider switching to the full version of jQuery while you are debugging the issue. You'll have a much easier time locating the line that is causing you trouble, which will often lead you in the direction of a solution.

If you're still stuck, consider posting your question to the jQuery Google group. Many of this book's authors are regular participants in the group, and more often than not, someone in the group will be able to offer useful advice. The `#jquery` IRC channel on Freenode is another valuable resource for troubleshooting issues.

If none of this works, it's possible we made a mistake. We worked hard to test and review all of the code in the book, but errors do creep through. Check the errata (described in the next section) and download the sample code, which will be updated to address any errata we discover.

If You Like (or Don't Like) This Book

If you like—or don't like—this book, by all means, please let people know. Amazon reviews are one popular way to share your happiness (or lack of happiness), or you can leave reviews at the site for the book:

http://oreilly.com/catalog/9780596159771/

There's also a link to errata there. Errata gives readers a way to let us know about typos, errors, and other problems with the book. That errata will be visible on the page immediately, and we'll confirm it after checking it out. O'Reilly can also fix errata in future printings of the book and on Safari, making for a better reader experience pretty quickly. We hope to keep this book updated for future versions of jQuery, and will also incorporate suggestions and complaints into future editions.

Conventions Used in This Book

The following typographical conventions are used in this book:

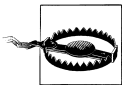
Italic
: Indicates Internet addresses, such as domain names and URLs, and new items where they are defined.

`Constant width`
: Indicates command lines and options that should be typed verbatim; names and keywords in programs, including method names, variable names, and class names; and HTML element tags, switches, attributes, keys, functions, types, namespaces, modules, properties, parameters, values, objects, events, event handlers, macros, the contents of files, or the output from commands.

`Constant width bold`
: Indicates emphasis in program code lines.

`Constant width italic`
: Indicates text that should be replaced with user-supplied values.

This icon signifies a tip, suggestion, or general note.



This icon indicates a warning or caution.

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 unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Answering a question by citing this book and quoting example code does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books *does* 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: "*jQuery Cookbook*, by Cody Lindley. Copyright 2010 Cody Lindley, 978-0-596-15977-1." If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Safari® Books Online

Safari Books Online is an on-demand digital library that lets you easily search over 7,500 technology and creative reference books and videos to find the answers you need quickly.

With a subscription, you can read any page and watch any video from our library online. Read books on your cell phone and mobile devices. Access new titles before they are available for print, and get exclusive access to manuscripts in development and post feedback for the authors. Copy and paste code samples, organize your favorites, download chapters, bookmark key sections, create notes, print out pages, and benefit from tons of other time-saving features.

O'Reilly Media has uploaded this book to the Safari Books Online service. To have full digital access to this book and others on similar topics from O'Reilly and other publishers, sign up for free at *http://my.safaribooksonline.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)

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 website at:

http://oreilly.com

—Rebecca Murphey and Cody Lindley

CHAPTER 1
jQuery Basics

Cody Lindley

1.0 Introduction

Since you've picked up a cookbook about jQuery, the authors of this book for the most part are going to assume that you have a loose idea about what exactly jQuery is and what it does. Frankly, cookbooks in general are typically written for an audience who seeks to enhance a foundation of knowledge that has already been established. Thus, the recipe-solution-discussion format is used to quickly get you solutions to common problems. However, if you are a jQuery newbie, don't throw this book against the wall and curse us just yet. We've dedicated this chapter to you.

If you are in need of a review or are jumping into this cookbook with little or no working knowledge of jQuery, this first chapter alone (the other chapters assume you know the basics) will aid you in learning the jQuery essentials. Now, realistically, if you have absolutely zero knowledge of JavaScript and the DOM, you might want to take a step back and ask yourself whether approaching jQuery *without* a basic understanding of the JavaScript core language and its relationship with the DOM is plausible. It would be my recommendation to study up on the DOM and JavaScript core before approaching jQuery. I highly recommend *JavaScript: The Definitive Guide* by David Flanagan (O'Reilly) as a primer before reading this book. But don't let my humble opinion stop you if you are attempting to learn jQuery before you learn about the DOM and JavaScript. Many have come to a working knowledge of these technologies by way of jQuery. And while not ideal, let's face it, it can still be done.

With that said, let's take a look at a formal definition of jQuery and a brief description of its functionality:

> jQuery is an open source JavaScript library that simplifies the interactions between an HTML document, or more precisely the Document Object Model (aka the DOM), and JavaScript.

In plain words, and for the old-school JavaScript hackers out there, jQuery makes Dynamic HTML (DHTML) dead easy. Specifically, jQuery simplifies HTML document

traversing and manipulation, browser event handling, DOM animations, Ajax interactions, and cross-browser JavaScript development.

With a formal explanation of jQuery under our belts, let's next explore why you might choose to use jQuery.

Why jQuery?

It might seem a bit silly to speak about the merits of jQuery within this cookbook, especially since you're reading this cookbook and are likely already aware of the merits.

So, while I might be preaching to the choir here, we're going to take a quick look at why a developer might choose to use jQuery. My point in doing this is to foster your basic knowledge of jQuery by first explaining the "why" before we look at the "how."

In building a case for jQuery, I'm not going to compare jQuery to its competitors in order to elevate jQuery's significance. That's because I just don't believe that there really is a direct competitor. Also, I believe the only library available today that meets the needs of both designer types and programmer types is jQuery. In this context, jQuery is in a class of its own.

Of the notorious JavaScript libraries and frameworks in the wild, I truly believe each has its own niche and value. A broad comparison is silly, but it's nevertheless attempted all the time. Heck, I am even guilty of it myself. However, after much thought on the topic, I truly believe that all JavaScript libraries are good at something. They all have value. What makes one more valuable than the other depends more upon who is using it and how it's being used than what it actually does. Besides, it has been my observation that micro differences across JavaScript libraries are often trivial in consideration of the broader goals of JavaScript development. So, without further philosophical ramblings, here is a list of attributes that builds a case for why you should use jQuery:

- It's open source, and the project is licensed under an MIT and a GNU General Public License (GPL) license. It's free, yo, in multiple ways!
- It's small (18 KB minified) and gzipped (114 KB, uncompressed).
- It's incredibly popular, which is to say it has a large community of users and a healthy amount of contributors who participate as developers and evangelists.
- It normalizes the differences between web browsers so that you don't have to.
- It's intentionally a lightweight footprint with a simple yet clever plugin architecture.
- Its repository of plugins is vast and has seen steady growth since jQuery's release.
- Its API is fully documented, including inline code examples, which in the world of JavaScript libraries is a luxury. Heck, any documentation at all was a luxury for years.
- It's friendly, which is to say it provides helpful ways to avoid conflicts with other JavaScript libraries.

- Its community support is actually fairly useful, including several mailing lists, IRC channels, and a freakishly insane amount of tutorials, articles, and blog posts from the jQuery community.
- It's openly developed, which means anyone can contribute bug fixes, enhancements, and development help.
- Its development is steady and consistent, which is to say the development team is not afraid of releasing updates.
- Its adoption by large organizations has and will continue to breed longevity and stability (e.g., Microsoft, Dell, Bank of America, Digg, CBS, Netflix).
- It's incorporating specifications from the W3C before the browsers do. As an example, jQuery supports a good majority of the CSS3 selectors.
- It's currently tested and optimized for development on modern browsers (Chrome 1, Chrome Nightly, IE 6, IE 7, IE 8, Opera 9.6, Safari 3.2, WebKit Nightly, Firefox 2, Firefox 3, Firefox Nightly).
- It's downright powerful in the hands of designer types as well as programmers. jQuery does not discriminate.
- Its elegance, methodologies, and philosophy of changing the way JavaScript is written is becoming a standard in and of itself. Consider just how many other solutions have borrowed the selector and chaining patterns.
- Its unexplainable by-product of feel-good programming is contagious and certainly unavoidable; even the critics seem to fall in love with aspects of jQuery.
- Its documentation has many outlets (e.g., API browser, dashboard apps, cheat sheets) including an offline API browser (AIR application).
- It's purposely bent to facilitate unobtrusive JavaScript practices.
- It has remained a JavaScript library (as opposed to a framework) at heart while at the same time providing a sister project for user interface widgets and application development (jQuery UI).
- Its learning curve is approachable because it builds upon concepts that most developers and designers already understand (e.g., CSS and HTML).

It is my opinion that the combination of the aforementioned jQuery points, and not any single attribute on its own, sets it apart from all other solutions. The total jQuery package is simply unmatched as a JavaScript tool.

The jQuery Philosophy

The jQuery philosophy is "Write less, do more." This philosophy can be further broken down into three concepts:

- Finding some elements (via CSS selectors) and doing something with them (via jQuery methods)
- Chaining multiple jQuery methods on a set of elements

- Using the jQuery wrapper and implicit iteration

Understanding these three concepts in detail is foundational when it comes time to write your own jQuery code or augment the recipes found in this book. Let's examine each of these concepts in detail.

Find some elements and do something with them

Or more specifically stated, locate a set of elements in the DOM, and then do something with that set of elements. For example, let's examine a scenario where you want to hide a `<div>` from the user, load some new text content into the hidden `<div>`, change an attribute of the selected `<div>`, and then finally make the hidden `<div>` visible again.

This last sentence translated into jQuery code would look something like this:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
</head>
<body>
<div>old content</div>
<script>

//hide all divs on the page
jQuery('div').hide();

//update the text contained inside of all divs
jQuery('div').text('new content');

//add a class attribute with a value of updatedContent to all divs
jQuery('div').addClass("updatedContent");

//show all divs on the page
jQuery('div').show();

</script>
</body>
</html>
```

Let's step through these four jQuery statements:

- Hide the `<div>` element on the page so it's hidden from the user's view.
- Replace the text inside the hidden `<div>` with some new text (`new content`).
- Update the `<div>` element with a new attribute (`class`) and value (`updatedContent`).
- Show the `<div>` element on the page so it's visible again to the viewing user.

If the jQuery code at this point is mystical syntax to you, that's OK. We'll dive into the basics with the first recipe in this chapter. Again, what you need to take away from this code example is the jQuery concept of "find some elements and do something with

them." In our code example, we found all the `<div>` elements in the HTML page using the jQuery function (`jQuery()`), and then using jQuery methods we did something with them (e.g., `hide()`, `text()`, `addClass()`, `show()`).

Chaining

jQuery is constructed in a manner that will allow jQuery methods to be chained. For example, why not find an element once and then chain operations onto that element? Our former code example demonstrating the "Find some elements and do something with them" concept could be rewritten to a single JavaScript statement using chaining.

This code, using chaining, can be changed from this:

```
//hide all divs on the page
jQuery('div').hide();

//update the text contained inside of the div
jQuery('div').text('new content');

//add a class attribute with a value of updatedContent to all divs
jQuery('div').addClass("updatedContent");

//show all divs on the page
jQuery('div').show();
```

to this:

```
jQuery('div').hide().text('new content').addClass("updatedContent").show();
```

or, with indenting and line breaks, to this:

```
jQuery('div')
    .hide()
    .text('new content')
    .addClass("updatedContent")
    .show();
```

Plainly speaking, chaining simply allows you to apply an endless chain of jQuery methods on the elements that are currently selected (currently wrapped with jQuery functionality) using the jQuery function. Behind the scenes, the elements previously selected before a jQuery method was applied are always returned so that the chain can continue. As you will see in future recipes, plugins are also constructed in this manner (returning wrapped elements) so that using a plugin does not break the chain.

If it's not immediately obvious, and based on the code in question, chaining also cuts down on processing overhead by selecting a set of DOM elements only once, to then be operated on numerous times by jQuery methods by way of chaining. Avoiding unnecessary DOM traversing is a critical part of page performance enhancements. Whenever possible, reuse or cache a set of selected DOM elements.

The jQuery wrapper set

A good majority of the time, if jQuery is involved, you're going to be getting what is known as a *wrapper*. In other words, you'll be selecting DOM elements from an HTML page that will be wrapped with jQuery functionality. Personally, I often refer to this as a "wrapper set" or "wrapped set" because it's a set of elements wrapped with jQuery functionality. Sometimes this wrapper set will contain one DOM element; other times it will contain several. There are even cases where the wrapper set will contain no elements. In these situations, the methods/properties that jQuery provides will fail silently if methods are called on an empty wrapper set, which can be handy in avoiding unneeded `if` statements.

Now, based on the code we used to demonstrate the "Find some elements and do something with them" concept, what do you think would happen if we added multiple `<div>` elements to the web page? In the following updated code example, I have added three additional `<div>` elements to the HTML page, for a total of four `<div>` elements:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<script type="text/JavaScript" src="http://ajax.googleapis.com/ajax/libs/
jquery/1.3.0/jquery.min.js"></script> </head>
<body>
<div>old content</div>
<div>old content</div>
<div>old content</div>
<div>old content</div>
<script>
//hide all divs on the page
jQuery('div').hide().text('new content').addClass("updatedContent").show();

</script>
</body>
</html>
```

You may not have explicitly written any programmatic loops here, but guess what? jQuery is going to scan the page and place all `<div>` elements in the wrapper set so that the jQuery methods I am using here are performed (aka implicit iteration) on each DOM element in the set. For example, the `.hide()` method actually applies to each element in the set. So if you look at our code again, you will see that each method that we use will be applied to each `<div>` element on the page. It's as if you had written a loop here to invoke each jQuery method on each DOM element. The updated code example will result in each `<div>` in the page being hidden, filled with updated text, given a new class value, and then made visible again.

Wrapping your head around (pun intended) the wrapper set and its default looping system (aka implicit iteration) is critical for building advanced concepts around looping. Just keep in mind that a simple loop is occurring here before you actually do any additional looping (e.g., `jQuery('div').each(function(){})`. Or another way to look at

this is each element in the wrapper will typically be changed by the jQuery method(s) that are called.

Something to keep in mind here is there are scenarios that you will learn about in the coming chapters where only the first element, and not all the elements in the wrapper set, is affected by the jQuery method (e.g., `attr()`).

How the jQuery API Is Organized

There is no question that when I first started out with jQuery, my main reason for selecting it as my JavaScript library was simply that it had been properly documented (and the gazillion plugins!). Later, I realized another factor that cemented my love affair with jQuery was the fact that the API was organized into logical categories. Just by looking at how the API was organized, I could narrow down the functionality I needed.

Before you really get started with jQuery, I suggest visiting the documentation online and simply digesting how the API is organized. By understanding how the API is organized, you'll more quickly navigate the documentation to the exact information you need, which is actually a significant advantage given that there are really a lot of different ways to code a jQuery solution. It's so robust that it's easy to get hung up on implementation because of the number of solutions for a single problem. I've replicated here for you how the API is organized. I suggest memorizing the API outline, or at the very least the top-level categories.

- jQuery Core
  - The jQuery Function
  - jQuery Object Accessors
  - Data
  - Plugins
  - Interoperability
- Selectors
  - Basics
  - Hierarchy
  - Basic Filters
  - Content Filters
  - Visibility Filters
  - Attribute Filters
  - Child Filters
  - Forms
  - Form Filters
- Attributes

    - Attr
    - Class
    - HTML
    - Text
    - Value
- Traversing
    - Filtering
    - Finding
    - Chaining
- Manipulation
    - Changing Contents
    - Inserting Inside
    - Inserting Outside
    - Inserting Around
    - Replacing
    - Removing
    - Copying
- CSS
    - CSS
    - Positioning
    - Height and Widths
- Events
    - Page Load
    - Event Handling
    - Live Events
    - Interaction Helpers
    - Event Helpers
- Effects
    - Basics
    - Sliding
    - Fading
    - Custom
    - Settings
- Ajax
    - AJAX Requests

  - —AJAX Events
  - —Misc.
- Utilities
  - —Browser and Feature Detection
  - —Array and Object Operations
  - —Test Operations
  - —String Operations
  - —Urls

Before we jump into a sequence of basic jQuery recipes, I would like to mention that the recipes found in this chapter build on each other. That is, there is a logical formation of knowledge as you progress from the first recipe to the last. It's my suggestion, for your first reading of these recipes, that you read them in order from 1.1 to 1.17.

1.1 Including the jQuery Library Code in an HTML Page

Problem

You want to use the jQuery JavaScript library on a web page.

Solution

There are currently two ideal solutions for embedding the jQuery library in a web page:

- Use the Google-hosted content delivery network (CDN) to include a version of jQuery (used in this chapter).
- Download your own version of jQuery from jQuery.com and host it on your own server or local filesystem.

Discussion

Including the jQuery JavaScript library isn't any different from including any other external JavaScript file. You simply use the HTML `<script>` element and provide the element a value (URL or directory path) for its `src=""` attribute, and the external file you are linking to will be included in the web page. For example, the following is a template that includes the jQuery library that you can use to start any jQuery project:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
</head>
<body>
```

```
<script type="text/JavaScript">
    alert('jQuery ' + jQuery.fn.jquery);
</script>
</body>
</html>
```

Notice that I am using—and highly recommend using for public web pages—the Google-hosted minified version of jQuery. However, debugging JavaScript errors in minified code is not ideal. During code development, or on the production site, it actually might be better to use the nonminified version from Google for the purpose of debugging potential JavaScript errors. For more information about using the Google-hosted version of jQuery, you can visit the Ajax libraries API site on the Web at *http://code.google.com/apis/ajaxlibs/*.

It's of course also possible, and mostly likely old hat, to host a copy of the jQuery code yourself. In most circumstances, however, this would be silly to do because Google has been kind enough to host it for you. By using a Google-hosted version of jQuery, you benefit from a stable, reliable, high-speed, and globally available copy of jQuery. As well, you reap the benefit of decreased latency, increased parallelism, and better caching. This of course could be accomplished without using Google's solution, but it would most likely cost you a dime or two.

Now, for whatever reason, you might not want to use the Google-hosted version of jQuery. You might want a customized version of jQuery, or your usage might not require/have access to an Internet connection. Or, you simply might believe that Google is "The Man" and wish not to submit to usage because you are a control freak and conspiracy fanatic. So, for those who do not need, or simply who do not want, to use a Google-hosted copy of the jQuery code, jQuery can be downloaded from jQuery.com and hosted locally on your own server or local filesystem. Based on the template I've provided in this recipe, you would simply replace the `src` attribute value with a URL or directory path to the location of the jQuery JavaScript file you've downloaded.

1.2 Executing jQuery/JavaScript Coded After the DOM Has Loaded but Before Complete Page Load

Problem

Modern JavaScript applications using unobtrusive JavaScript methodologies typically execute JavaScript code only after the DOM has been completely loaded. And the reality of the situation is that any DOM traversing and manipulation will require that the DOM is loaded before it can be operated on. What's needed is a way to determine when the client, most often a web browser, has completely loaded the DOM but has possibly not yet completely loaded all assets such as images and SWF files. If we were to use the `window.onload` event in this situation, the entire document including all assets would

need to be completely loaded before the `onload` event fired. That's just too time-consuming for most web surfers. What's needed is an event that will tell us when the DOM alone is ready to be traversed and manipulated.

Solution

jQuery provides the `ready()` method, which is a custom event handler that is typically bound to the DOM's document object. The `ready()` method is passed a single parameter, a function, that contains the JavaScript code that should be executed once the DOM is ready to be traversed and manipulated. The following is a simple example of this event opening an `alert()` window once the DOM is ready but before the page is completely loaded:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
    jQuery(document).ready(function(){//DOM not loaded, must use ready event
        alert(jQuery('p').text());
    });
</script>
</head>
<body>
<p>The DOM is ready!</p>
</body>
</html>
```

Discussion

The `ready()` event handler method is jQuery's replacement for using the JavaScript core `window.onload` event. It can be used as many times as you like. When using this custom event, it's advisable that it be included in your web pages after the inclusion of stylesheet declarations and includes. Doing this will ensure that all element properties are correctly defined before any jQuery code or JavaScript code will be executed by the `ready()` event.

Additionally, the jQuery function itself provides a shortcut for using the jQuery custom ready event. Using this shortcut, the following `alert()` example can be rewritten like so:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
    jQuery(function(){ //DOM not loaded, must use ready event
```

```
        alert(jQuery('p').text());
    });
</script>
</head>
<body>
<p>The DOM is ready!</p>
</body>
</html>
```

The use of this custom jQuery event is necessary only if JavaScript has to be embedded in the document flow at the top of the page and encapsulated in the `<head>` element. I simply avoid the usage of the `ready()` event by placing all JavaScript includes and inline code before the closing `<body>` element. I do this for two reasons.

First, modern optimization techniques have declared that pages load faster when the JavaScript is loaded by the browser at the end of a page parse. In other words, if you put JavaScript code at the bottom of a web page, then the browser will load everything in front of it before it loads the JavaScript. This is a good thing because most browsers will typically stop processing other loading initiatives until the JavaScript engine has compiled the JavaScript contained in a web page. It's sort of a bottleneck in a sense that you have JavaScript at the top of a web page document. I realize that for some situations it's easier to place JavaScript in the `<head>` element. But honestly, I've never seen a situation where this is absolutely required. Any obstacle that I've encountered during my development by placing JavaScript at the bottom of the page has been easily overcome and well worth the optimization gains.

Second, if speedy web pages are our goal, why wrap more functionality around a situation that can be elevated by simply moving the code to the bottom of the page? When given the choice between more code or less code, I choose less code. Not using the `ready()` event results in using less code, especially since less code always runs faster than more code.

With some rationale out of the way, here is an example of our `alert()` code that does not use the `ready()` event:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<p>The DOM is ready!</p>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
     alert(jQuery('p').text());//go for it the DOM is loaded
</script>
</body>
</html>
```

Notice that I have placed all of my JavaScript before the closing `</body>` element. Any additional markup should be placed above the JavaScript in the HTML document.

1.3 Selecting DOM Elements Using Selectors and the jQuery Function

Problem

You need to select a single DOM element and/or a set of DOM elements in order to operate on the element(s) using jQuery methods.

Solution

jQuery provides two options when you need to select element(s) from the DOM. Both options require the use of the jQuery function (`jQuery()` or alias `$()`). The first option, which uses CSS selectors and custom selectors, is by far the most used and most eloquent solution. By passing the jQuery function a string containing a selector expression, the function will traverse the DOM and locate the DOM nodes defined by the expression. As an example, the following code will select all the `<a>` elements in the HTML document:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<a href='#'>link</a>
<a href='#'>link</a>
<a href='#'>link</a>
<a href='#'>link</a>
<a href='#'>link</a>
<a href='#'>link</a>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
     //alerts there are 6 elements
     alert('Page contains ' + jQuery('a').length +  ' <a> elements!');
</script>
</body>
</html>
```

If you were to run this HTML page in a web browser, you would see that the code executes a browser `alert()` that informs us that the page contains six `<a>` elements. I passed this value to the `alert()` method by first selecting all the `<a>` elements and then using the `length` property to return the number of elements in the jQuery wrapper set.

You should be aware that the first parameter of the jQuery function, as we are using it here, will also accept multiple expressions. To do this, simply separate multiple selectors with a comma inside the same string that is passed as the first parameter to the jQuery function. Here is an example of what that might look like:

```
jQuery('selector1, selector2, selector3').length;
```

Our second option for selecting DOM elements and the less common option is to pass the jQuery function an actual JavaScript reference to DOM element(s). As an example, the following code will select all the `<a>` elements in the HTML document. Notice that I'm passing the jQuery function an array of `<a>` elements collected using the `getElementsByTagName` DOM method. This example produces the same exact results as our previous code example:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body bgcolor="yellow"> <!-- yes the attribute is depreciated, I know, roll
with it -->
<a href='#'>link</a>
<a href='#'>link</a>
<a href='#'>link</a>
<a href='#'>link</a>
<a href='#'>link</a>
<a href='#'>link</a>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
     //alerts there are 6 elements
     alert('Page contains ' + jQuery(document.getElementsByTagName('a')).length +
' <a> Elements, And has a '
     + jQuery(document.body).attr('bgcolor') + ' background');
</script>
</body>
</html>
```

Discussion

The heavy lifting that jQuery is known for is partially based on the selector engine, Sizzle, that selects DOM element(s) from an HTML document. While you have the option, and it's a nice option when you need it, passing the jQuery function DOM references is not what put jQuery on everyone's radar. It's the vast and powerful options available with selectors that make jQuery so unique.

Throughout the rest of the book, you will find powerful and robust selectors. When you see one, make sure you fully understand its function. This knowledge will serve you well with future coding endeavors using jQuery.

1.4 Selecting DOM Elements Within a Specified Context

Problem

You need a reference to a single DOM element or a set of DOM elements in the context of another DOM element or document in order to operate on the element(s) using jQuery methods.

Solution

The jQuery function when passed a CSS expression will also accept a second parameter that tells the jQuery function to which context it should search for the DOM elements based on the expression. The second parameter in this case can be a DOM reference, jQuery wrapper, or document. In the following code, there are 12 `<input>` elements. Notice how I use a specific context, based on the `<form>` element, to select only particular `<input>` elements:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>

<form>
<input name="" type="checkbox" />
<input name="" type="radio" />
<input name="" type="text" />
<input name="" type="button" />
</form>

<form>
<input name="" type="checkbox" />
<input name="" type="radio" />
<input name="" type="text" />
<input name="" type="button" />
</form>

<input name="" type="checkbox" />
<input name="" type="radio" />
<input name="" type="text" />
<input name="" type="button" />

<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">

 //searches within all form elements, using a wrapper for context, alerts "8 inputs"
 alert('selected ' + jQuery('input',$('form')).length + ' inputs');

 //search with the first form element, using DOM reference as the context, alerts
```

```
    //"4 inputs"
    alert('selected' + jQuery('input',document.forms[0]).length + ' inputs');

    //search within the body element for all input elements using an expression,
    //alerts "12 inputs"
    alert('selected' + jQuery('input','body').length + ' inputs');

</script>
</body>
</html>
```

Discussion

It's also possible, as mentioned in the solution of this recipe, to select documents as the context for searching. For example, it's possible to search within the context of an XML document that is sent back from doing an XHR request (Ajax). You can find more details about this usage in Chapter 16.

1.5 Filtering a Wrapper Set of DOM Elements

Problem

You have a set of selected DOM elements in a jQuery wrapper set but want to remove DOM elements from the set that do not match a new specified expression(s) in order to create a new set of elements to operate on.

Solution

The jQuery filter method, used on a jQuery wrapper set of DOM elements, can exclude elements that *do not* match a specified expression(s). In short, the `filter()` method allows you to filter the current set of elements. This is an important distinction from the jQuery find method, which will reduce a wrapped set of DOM elements by finding (via a new selector expression) new elements, including child elements of the current wrapped set.

To understand the filter method, let's examine the following code:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<a href="#" class="external">link</a>
<a href="#" class="external">link</a>
<a href="#"></a>
<a href="#" class="external">link</a>
<a href="#" class="external">link</a>
<a href="#"></a></li>
```

```
<a href="#">link</a>
<a href="#">link</a>
<a href="#">link</a>
<a href="#">link</a>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">

   //alerts 4 left in the set
       alert(jQuery('a').filter('.external').length + ' external links');
</script>
</body>
</html>
```

The HTML page in the code example just shown contains a web page with 10 `<a>` elements. Those links that are external links are given a class name of `external`. Using the jQuery function, we select all `<a>` elements on the page. Then, using the filter method, all those elements that do not have a `class` attribute value of `external` are removed from the original set. Once the initial set of DOM elements are altered using the `filter()` method, I invoke the `length` property, which will tell me how many elements are now in my new set after the filter has been applied.

Discussion

It's also possible to send the `filter()` method a function that can be used to filter the wrapped set. Our previous code example, which passes the `filter()` method a string expression, can be changed to use a function instead:

```
alert(
   jQuery('a')
      .filter(function(index){ return $(this).hasClass('external');})
      .length + ' external links'
);
```

Notice that I am now passing the `filter()` method an anonymous function. This function is called with a context equal to the current element. That means when I use `$(this)` within the function, I am actually referring to each DOM element in the wrapper set. Within the function, I am checking each `<a>` element in the wrapper set to see whether the element has a class value (`hasClass()`) of `external`. If it does, Boolean true, then keep the element in the set, and if it doesn't (false), then remove the element from the set. Another way to look at this is if the function returns false, then the element is removed. If the function returns any other data value besides false, then the element will remain in the wrapper set.

You may have noticed that I have passed the function a parameter named `index` that I am not using. This parameter, if needed, can be used to refer numerically to the index of the element in the jQuery wrapper set.

1.6 Finding Descendant Elements Within the Currently Selected Wrapper Set

Problem

You have a set of selected DOM elements (or a single element) and want to find descendant (children) elements within the context of the currently selected elements.

Solution

Use the `.find()` method to create a new wrapper set of elements based on the context of the current set and their descendants. For example, say that you have a web page that contains several paragraphs. Encapsulated inside of these paragraphs are words that are emphasized (italic). If you'd like to select only `<em>` elements contained within `<p>` elements, you could do so like this:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<p>Ut ad videntur facilisis <em>elit</em> cum. Nibh insitam erat facit
<em>saepius</em> magna.  Nam ex liber iriure et imperdiet. Et mirum eros
iis te habent. </p>
<p>Claram claritatem eu amet dignissim magna. Dignissim quam elit facer eros
illum. Et qui ex esse <em>tincidunt</em> anteposuerit. Nulla nam odio ii
vulputate feugait.</p>
<p>In quis <em>laoreet</em> te legunt euismod. Claritatem <em>consuetudium</em>
wisi sit velit facilisi.</p>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
 //alerts total italic words found inside of <p> elements
       alert('The three paragraphs in all contain ' +
       jQuery('p').find('em').length + '
italic words');
</script>
</body>
</html>
```

Keep in mind that we could have also written this code by passing a contextual reference as a second parameter to the jQuery function:

```
alert('The three paragraphs in all contain ' + jQuery('em',$('p')).length +
' italic words');
```

Additionally, it's worth mentioning that the last two code examples are demonstrative in purpose. It is likely more logical, if not pragmatic, to use a CSS selector expression to select all the descendant italic elements contained within the ancestor `<p>` elements.

```
alert('The three paragraphs in all contain ' + jQuery('p em').length +
' italic words');
```

Discussion

The jQuery `.find()` method can be used to create a new set of elements based on context of the current set of DOM elements and their children elements. People often confuse the use of the `.filter()` method and `.find()` method. The easiest way to remember the difference is to keep in mind that `.find()` will operate/select the children of the current set while `.filter()` will only operate on the current set of elements. In other words, if you want to change the current wrapper set by using it as a context to further select the children of the elements selected, use `.find()`. If you only want to filter the current wrapped set and get a new subset of the current DOM elements in the set only, use `.filter()`. To boil this down even more, `find()` returns children elements, while `filter()` only filters what is in the current wrapper set.

1.7 Returning to the Prior Selection Before a Destructive Change

Problem

A destructive jQuery method (e.g., `filter()` or `find()`) that was used on a set of elements needs to be removed so that the set prior to the use of the destructive method is returned to its previous state and can then be operated as if the destructive method had never been invoked.

Solution

jQuery provides the `end()` method so that you can return to the previous set of DOM elements that were selected before using a destructive method. To understand the `end()` method, let's examine the following HTML.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<p>text</p>
<p class="middle">Middle <span>text</span></p>
<p>text</p>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
    alert(jQuery('p').filter('.middle').length); //alerts 1
    alert(jQuery('p').filter('.middle').end().length); //alerts 3
    alert(jQuery('p').filter('.middle').find('span')
```

```
.end().end().length); //alerts 3
</script>
</body>
</html>
```

The first `alert()` statement in the code contains a jQuery statement that will search the document for all `<p>` elements and then apply `filter()` to the selected `<p>` elements in the set selecting only the one(s) with a class of `middle`. The `length` property then reports how many elements are left in the set:

```
alert(jQuery('p').filter('.middle').length); //alerts 1
```

The next `alert()` statement makes use of the `end()` method. Here we are doing everything we did in the prior statement except that we are undoing the `filter()` method and returning to the set of elements contained in the wrapper set before the `filter()` method was applied:

```
alert(jQuery('p').filter('.middle').end().length); //alerts 3
```

The last `alert()` statement demonstrates how the `end()` method is used twice to remove both the `filter()` and `find()` destructive changes, returning the wrapper set to its original composition:

```
alert(jQuery('p').filter('.middle').find('span').end().end().length); //alerts 3
```

Discussion

If the `end()` method is used and there were no prior destructive operations performed, an empty set is returned. A destructive operation is any operation that changes the set of matched jQuery elements, which means any traversing or manipulation method that returns a jQuery object, including `add()`, `andSelf()`, `children()`, `closes()`, `filter()`, `find()`, `map()`, `next()`, `nextAll()`, `not()`, `parent()`, `parents()`, `prev()`, `prevAll()`, `siblings()`, `slice()`, `clone()`, `appendTo()`, `prependTo()`, `insertBefore()`, `insertAfter()`, and `replaceAll()`.

1.8 Including the Previous Selection with the Current Selection

Problem

You have just manipulated a set of elements in order to acquire a new set of elements. However, you want to operate on the prior set as well as the current set.

Solution

You can combine a prior selection of DOM elements with the current selection by using the `andSelf()` method. For example, in the following code, we are first selecting all `<div>` elements on the page. Next we manipulate this set of elements by finding all `<p>` elements contained within the `<div>` elements. Now, in order to operate on both the `<div>` and the `<p>` elements found within the `<div>`, we could include the `<div>` into

the current set by using `andSelf()`. Had I omitted the `andSelf()`, the border color would have only been applied to the `<p>` elements:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<div>
<p>Paragraph</p>
<p>Paragraph</p>
</div>
<script type="text/JavaScript" src="http://ajax.googleapis.com/
ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
    jQuery('div').find('p').andSelf().css('border','1px solid #993300');
</script>
</body>
</html>
```

Discussion

Keep in mind that when you use the `andSelf()` method, it will only add into the current set being operated on and the prior set, but not all prior sets.

1.9 Traversing the DOM Based on Your Current Context to Acquire a New Set of DOM Elements

Problem

You have selected a set of DOM elements, and based on the position of the selections within the DOM tree structure, you want to traverse the DOM to acquire a new set of elements to operate on.

Solution

jQuery provides a set of methods for traversing the DOM based on the context of the currently selected DOM element(s).

For example, let's examine the following HTML snippet:

```
<div>
<ul>
<li><a href="#">link</a></li>
<li><a href="#">link</a></li>
<li><a href="#">link</a></li>
<li><a href="#">link</a></li>
</ul>
</div>
```

Now, let's select the second `<li>` element using the `:eq()` index custom selector:

```
//selects the second element in the set of <li>'s by index, index starts at 0
jQuery('li:eq(1)');
```

We now have a context, a starting point within the HTML structure. Our starting point is the second `<li>` element. From here we can go anywhere—well, almost anywhere. Let's see where we can go using a couple of the methods jQuery provides for traversing the DOM. Read the comments in the code for clarification:

```
jQuery('li:eq(1)').next() //selects the third <li>

jQuery('li:eq(1)').prev() //selects the first <li>

jQuery('li:eq(1)').parent() //selects the <ul>

jQuery('li:eq(1)').parent().children() //selects all <li>s

jQuery('li:eq(1)').nextAll() //selects all the <li>s after the second <li>

jQuery('li:eq(1)').prevAll() //selects all the <li>s before the second <li>
```

Keep in mind that these traversing methods produce a new wrapper set, and to return to the previous wrapper set, you can use `end()`.

Discussion

The traversing methods shown thus far have demonstrated simple traverses. There are two additional concepts that are important to know about traversing.

The first concept and likely most obvious is that traversing methods can be chained. Let's examine again the following jQuery statement from earlier:

```
jQuery('li:eq(1)').parent().children() //selects all <li>'s
```

Notice that I have traversed from the second `<li>` element to the parent `<ul>` element and then again traversed from the parent element to selecting all the children elements of the `<ul>` element. The jQuery wrapper set will now contain all the `<li>` elements contained within the `<ul>`. Of course, this is a contrived example for the purpose of demonstrating traversing methods. Had we really wanted a wrapper set of just `<li>` elements, it would have been much simpler to select all the `<li>` elements from the get-go (e.g., `jQuery('li')`).

The second concept that you need to keep in mind when dealing with the traversing methods is that many of the methods will accept an optional parameter that can be used to filter the selections. Let's take our chained example again and look at how we could change it so that only the last `<li>` element was selected. Keep in mind that this is a contrived example for the purpose of demonstrating how a traversing method can be passed an expression used for selecting a very specific element:

```
jQuery('li:eq(1)').parent().children(':last') //selects the last <li>
```

jQuery provides additional traversing methods that were not shown here. For a complete list and documentation, have a look at *http://docs.jquery.com/Traversing*. You will find these additional traversing methods used throughout this book.

1.10 Creating, Operating on, and Inserting DOM Elements

Problem

You want to create new DOM elements (or a single element) that are immediately selected, operated on, and then injected into the DOM.

Solution

If you haven't figured it out yet, the jQuery function is multifaceted in that this one function performs differently depending upon the makeup of the parameter(s) you send it. If you provide the function with a text string of raw HTML, it will create these elements for you on the fly. For example, the following statement will create an `<a>` element wrapped inside of a `<p>` element with a text node encapsulated inside of the `<p>` and `<a>` elements:

```
jQuery('<p><a>jQuery</a></p>');
```

Now, with an element created, you can use jQuery methods to further operate on the elements you just created. It's as if you had selected the `<p>` element from the get-go in an existing HTML document. For example, we could operate on the `<a>` by using the `.find()` method to select the `<a>` element and then set one of its attributes. In the case of the following code, we are setting the `href` attribute with a value of `http://www.jquery.com`:

```
jQuery('<p><a>jQuery</a></p>').find('a').attr('href','http://www.jquery.com');
```

This is great, right? Well, it's about to get better because all we have done so far is create elements on the fly and manipulate those elements in code. We have yet to actually change the currently loaded DOM, so to speak. To do this, we'll have to use the manipulation methods provided by jQuery. The following is our code in the context of an HTML document. Here we are creating elements, operating on those elements, and then inserting those elements into the DOM using the `appendTo()` manipulation method:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
```

```
jQuery('<p><a>jQuery</a></p>').find('a').attr('href','http://www.jquery.com')
  .end().appendTo('body');
</script>
</body>
</html>
```

Notice how I am using the `end()` method here to undo the `find()` method so that when I call the `appendTo()` method, it appends what was originally contained in the initial wrapper set.

Discussion

In this recipe we've passed the jQuery function a string of raw HTML that is taken and used to create DOM elements on the fly. It's also possible to simply pass the jQuery function a DOM object created by the DOM method `createElement()`:

```
jQuery(document.createElement('p')).appendTo('body'); //adds an empty p element
to the page
```

Of course, this could be rather laborious depending upon the specifics of the usage when a string of HTML containing multiple elements will work just fine.

It's also worth mentioning here that we've only scratched the surface of the manipulation methods by using the `appendTo()` method. In addition to the `appendTo()` method, there are also the following manipulation methods:

- `append()`
- `prepend()`
- `prependTo()`
- `after()`
- `before()`
- `insertAfter()`
- `insertBefore()`
- `wrap()`
- `wrapAll()`
- `wrapInner()`

1.11 Removing DOM Elements

Problem

You want to remove elements from the DOM.

Solution

The `remove()` method can be used to remove a selected set of elements and their children elements from the DOM. Examine the following code:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<h3>Anchors</h3>
<a href='#'>Anchor Element</a>
<a href='#'>Anchor Element</a>
<a href='#'>Anchor Element</a>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
  jQuery('a').remove();
</script>
</body>
</html>
```

When the preceding code is loaded into a browser, the anchor elements will remain in the page until the JavaScript is executed. Once the `remove()` method is used to remove all anchor elements from the DOM, the page will visually contain only an `<h3>` element.

It's also possible to pass the method an expression to filter the set of elements to be removed. For example, our code could change to remove only anchors with a specific class:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<h3>Anchors</h3>
<a href='#' class='remove'>Anchor Element</a>
<a href='#'>Anchor Element</a>
<a href='#' class="remove">Anchor Element</a>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
  jQuery('a').remove('.remove');
</script>
</body>
</html>
```

Discussion

When using the jQuery `remove()` method, you need to keep two things in mind:

- While the elements selected are removed from the DOM using `remove()`, they have not been removed from the jQuery wrapper set. That means in theory you could continue operating on them and even add them back into the DOM if desired.
- This method will not only remove the elements from the DOM, but it will also remove all event handlers and internally cached data that the elements removed might have contained.

1.12 Replacing DOM Elements

Problem

You need to replace DOM nodes currently in the DOM with new DOM nodes.

Solution

Using the `replaceWith()` method, we can select a set of DOM elements for replacement. In the following code example, we use the `replaceWith()` method to replace all `<li>` elements with a `class` attribute of `remove` with a new DOM structure:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<ul>
<li class='remove'>name</li>
<li>name</li>
<li class='remove'>name</li>
<li class='remove'>name</li>
<li>name</li>
<li class='remove'>name</li>
<li>name</li>
<li class='remove'>name</li>
</ul>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
  jQuery('li.remove').replaceWith('<li>removed</li>');
</script>
</body>
</html>
```

The new DOM structure added to the DOM is a string parameter passed into the `replaceWith()` method. In our example, all the `<li>` elements, including children elements, are replaced with the new structure, `<li>removed</li>`.

Discussion

jQuery provides an inverse to this method called `replaceAll()` that does the same task with the parameters reversed. For example, we could rewrite the jQuery code found in our recipe code like so:

```
jQuery('<li>removed</li>').replaceAll('li.remove');
```

Here we are passing the jQuery function the HTML string and then using the `replaceAll()` method to select the DOM node and its children that we want to be removed and replaced.

1.13 Cloning DOM Elements

Problem

You need to clone/copy a portion of the DOM.

Solution

jQuery provides the `clone()` method for copying DOM elements. Its usage is straightforward. Simply select the DOM elements using the jQuery function, and then call the `clone()` method on the selected set of element(s). The result is a copy of the DOM structure being returned for chaining instead of the originally selected DOM elements. In the following code, I am cloning the `<ul>` element and then appending this copy back into the DOM using the inserting method `appendTo()`. Essentially, I am adding another `<ul>` structure to the page exactly like the one that is already there:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<ul>
<li>list</li>
<li>list</li>
<li>list</li>
<li>list</li>
</ul>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
 jQuery('ul').clone().appendTo('body');
</script>
</body>
</html>
```

Discussion

The cloning method is actually very handy for moving DOM snippets around inside of the DOM. It's especially useful when you want to not only copy and move the DOM elements but also the events attached to the cloned DOM elements. Closely examine the HTML and jQuery here:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<ul id="a">
<li>list</li>
<li>list</li>
<li>list</li>
<li>list</li>
</ul>
<ul id="b"></ul>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script type="text/JavaScript">
 jQuery('ul#a li')
    .click(function(){alert('List Item Clicked')})
    .parent()
        .clone(true)
            .find('li')
            .appendTo('#b')
        .end()
    .end()
    .remove();
</script>
</body>
</html>
```

If you were to run this code in a browser, it would clone the `<li>` elements on the page that have a click event attached to them, insert these newly cloned elements (including events) into the empty `<ul>`, and then remove the `<ul>` element that we cloned.

This might stretch a new jQuery developer's mind, so let's examine this jQuery statement by stepping through this code in order to explain the chained methods:

1. `jQuery('ul#a li')` = Select `<ul>` element with an `id` attribute of `a` and then select all the `<li>` elements inside of the `<ul>`.
2. `.click(function(){alert('List Item Clicked')})` = Add a click event to each `<li>`.
3. `.parent()` = Traverse the DOM, by changing my selected set to the `<ul>` element.
4. `.clone(true)` = Clone the `<ul>` element and all its children, including any events attached to the elements that are being cloned. This is done by passing the `clone()` method a Boolean value of `true`.

5. `.find('li')` = Now, within the cloned elements, change the set of elements to only the `<li>` elements contained within the cloned `<ul>` element.
6. `.appendTo('#b')` = Take these selected cloned `<li>` elements and place them inside of the `<ul>` element that has an `id` attribute value of `b`.
7. `.end()` = Return to the previous selected set of elements, which was the cloned `<ul>` element.
8. `.end()` = Return to the previous selected set of elements, which was the original `<ul>` element we cloned.
9. `.remove()` = Remove the original `<ul>` element.

If it's not obvious, understanding how to manipulate the selected set of elements or revert to the previous selected set is crucial for complex jQuery statements.

1.14 Getting, Setting, and Removing DOM Element Attributes

Problem

You have selected a DOM element using the jQuery function and need to get or set the value of the DOM element's attribute.

Solution

jQuery provides the `attr()` method for getting and setting attribute values. In the following code, we are going to be setting and then getting the value of an `<a>` element's `href` attribute:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<a>jquery.com</a>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js">
</script>
<script type="text/JavaScript">
// alerts the jQuery home page URL
alert(
    jQuery('a').attr('href','http://www.jquery.com').attr('href')
);
</script>
</body>
</html>
```

As you can see in the code example, we are selecting the only `<a>` element in the HTML document, setting its `href` attribute, and then getting its value with the same `attr()`

method by passing the method the attribute name alone. Had there been multiple `<a>` elements in the document, the `attr()` method would access the first matched element. The code when loaded into a browser will `alert()` the value that we set for the `href` attribute.

Now, since most elements have more than one attribute available, it's also possible to set multiple attribute values using a single `attr()` method. For example, we could also set the `title` attribute in the previous example by passing the `attr()` method an object instead of two string parameters:

```
jQuery('a').attr({'href':'http://www.jquery.com','title':'jquery.com'}).attr('href')
```

With the ability to add attributes to elements also comes the ability to remove attributes and their values. The `removeAttr()` method can be used to remove attributes from HTML elements. To use this method, simply pass it a string value of the attribute you'd like to remove (e.g., `jQuery('a')removeAttr('title')`).

Discussion

In addition to the `attr()` method, jQuery provides a very specific set of methods for working with the HTML element `class` attribute. Since the `class` attribute can contain several values (e.g., `class="class1 class2 class3"`), these unique attribute methods are used to manage these values.

These jQuery methods are as follows:

`addClass()`
: Updates the `class` attribute value with a new class/value including any classes that were already set

`hasClass()`
: Checks the value of the `class` attribute for a specific class

`removeClass()`
: Removes a unique class from the `class` attribute while keeping any values already set

`toggleClass()`
: Adds the specified class if it is not present; removes the specified class if it is present

1.15 Getting and Setting HTML Content

Problem

You need to get or set a chunk of HTML content in the current web page.

Solution

jQuery provides the `html()` method for getting and setting chunks (or DOM structures) of HTML elements. In the following code, we use this method to set and then get the HTML value of the `<p>` element found in the HTML document:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<p></p>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js">
</script>
<script type="text/JavaScript">
jQuery('p').html('<strong>Hello World</strong>, I am a <em>&lt;p&gt;</em> element.');
alert(jQuery('p').html());
</script>
</body>
</html>
```

Running this code in a browser will result in a browser alerting the HTML content contained within the `<p>` element, which we set using the `html()` method and then retrieved using the `html()` method.

Discussion

This method uses the DOM `innerHTML` property to get and set chunks of HTML. You should also be aware that `html()` is not available on XML documents (although it will work for XHTML documents).

1.16 Getting and Setting Text Content

Problem

You need to get or set the text that is contained inside of an HTML element(s).

Solution

jQuery provides the `text()` method for getting and setting the text content of elements. In the following code, we use this method to set and then get the text value of the `<p>` element found in the HTML document:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

```
</head>
<body>
<p></p>
<script type="text/JavaScript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js">
</script>
<script type="text/JavaScript">
     jQuery('p').text('Hello World, I am a <p> element.');
     alert(jQuery('p').text());
</script>
</body>
</html>
```

Running this code in a browser will result in a browser alerting the content of the `<p>` element, which we set using the `text()` method and then retrieved using the `text()` method.

Discussion

It's important to remember that the `text()` method is not unlike `html()` except that the `text()` method will escape HTML (replace `<` and `>` with their HTML entities). This means that if you place tags inside of the text string passed to the `text()` method, it will convert these tags to their HTML entities (`&lt;` and `&gt;`).

1.17 Using the $ Alias Without Creating Global Conflicts

Problem

You want to use the shortcut `$` alias instead of typing the global namespace name (jQuery) without fear of global conflicts.

Solution

The solution here is to create an anonymous self-invoking function that we pass the jQuery object to and then use the `$` character as a parameter pointer to the jQuery object.

For example, all jQuery code could be encapsulated inside the following self-invoking function:

```
(function($){ //function to create private scope with $ parameter
    //private scope and using $ without worry of conflict
})(jQuery); //invoke nameless function and pass it the jQuery object
```

Discussion

Essentially, what is going on here is that we have passed the global reference to jQuery to a function that creates a private scope. Had we not done this and chosen to use the shorthand `$` alias in the global scope, we would be taking a risk by assuming that no

other scripts included in the HTML document (or scripts included in the future) use the `$` character. Why risk it when you can just create your own private scope?

Another advantage to doing this is that code included inside of the anonymous self-invoking function will run in its own private scope. You can rest assured that anything that is placed inside the function will likely never cause a conflict with other JavaScript code written in the global scope. So, again, why risk programmatic collisions? Just create your own private scope.

CHAPTER 2

Selecting Elements with jQuery

James Padolsey

2.0 Introduction

At the very core of jQuery is its selector engine, allowing you to select elements within any document based on names, attributes, states, and more. Because of CSS's popularity, it made sense to adopt its selector syntax to make it simple to select elements in jQuery. As well as supporting most of the selectors specified in the CSS 1–3 specifications, jQuery adds quite a few *custom selectors* that can be used to select elements based on special states and characteristics. Additionally, you can create your own custom selectors! This chapter discusses some of the more common problems encountered while selecting elements with jQuery.

Before the first recipe, let's discuss a few basic principles.

The easiest way to target a specific element or a set of elements within a document is by using a CSS selector within the jQuery wrapper function, like so:

```
jQuery('#content p a');
    // Select all anchor elements within all paragraph elements within #content
```

Now that we've selected the elements we're after, we can run any of jQuery's methods on that collection. For example, adding a class of `selected` to all links is as simple as:

```
jQuery('#content p a').addClass('selected');
```

jQuery offers many DOM traversal methods to aid in the element selection process, such as `next()`, `prev()`, and `parent()`. These and other methods accept a selector expression as their only parameter, which filters the returned results accordingly. So, you can use CSS selectors in a number of places, not just within `jQuery(...)`.

When constructing selectors, there's one general rule for optimization: be only as specific as you need to be. It's important to remember that the more complicated a selector is, the more time it will take jQuery to process the string. jQuery uses native DOM methods to retrieve the elements you're after. The fact that you can use selectors is only a product of a nicely polished abstraction; there's nothing wrong with this, but it is

very important to understand the ramifications of what you're writing. Here is a typical example of an unnecessarily complicated selector:

```
jQuery('body div#wrapper div#content');
```

A higher degree of specificity does not necessarily mean it's faster. The previous selector can be rewritten to this:

```
jQuery('#content');
```

This has the same effect but manages to shave off the overhead of the previous version. Also note that sometimes you can further optimize by specifying a context for your selectors; this will be discussed later in the chapter (see Recipe 2.11).

2.1 Selecting Child Elements Only

Problem

You need to select one or more direct children of a particular element.

Solution

Use the *direct descendant* combinator (>). This combinator expects two selector expressions, one on either side. For example, if you want to select all anchor elements that reside directly beneath list items, you could use this selector: `li > a`. This would select all anchors that are children of a list item; in other words, all anchors that exist directly beneath list items. Here's an example:

```
<a href="/category">Category</a>
<ul id="nav">
    <li><a href="#anchor1">Anchor 1</a></li>
    <li><a href="#anchor2">Anchor 2</a></li>
    <li><span><a href="#anchor3">Anchor 3</a></span></li>
</ul>
```

Now, to select only the anchors within each list item, you would call jQuery like so:

```
jQuery('#nav li > a');
 // This selects two elements, as expected
```

The third anchor within the `#nav` list is not selected because it's not a child of a list item; it's a child of a `<span>` element.

Discussion

It's important to distinguish between a child and a descendant. A *descendant* is any element existing within another element, whereas a *child* is a direct descendant; the analogy of children and parents helps massively since the DOM's hierarchy is largely similar to that.

It's worth noting that combinators like > can be used without an expression on the left side if a context is already specified:

```
jQuery('> p', '#content');
    // Fundamentally the same as jQuery('#content > p')
```

Selecting children in a more programmatic environment should be done using jQuery's `children()` method, to which you can pass a selector to filter the returned elements. This would select all direct children of the `#content` element:

```
jQuery('#content').children();
```

The preceding code is essentially the same as `jQuery('#content > *')` with one important difference; it's faster. Instead of parsing your selector, jQuery knows what you want immediately. The fact that it's faster is not a useful differential, though. Plus, in some situations, the speed difference is marginal verging on irrelevant, depending on the browser and what you're trying to select. Using the `children()` method is especially useful when you're dealing with jQuery objects stored under variables. For example:

```
var anchors = jQuery('a');

// Getting all direct children of all anchor elements
// can be achieved in three ways:

// #1
anchors.children();

// #2
jQuery('> *', anchors);

// #3
anchors.find('> *');
```

In fact, there are even more ways of achieving it! In this situation, the first method is the fastest. As stated earlier, you can pass a selector expression to the `children()` method to filter the results:

```
jQuery('#content').children('p');
```

Only paragraph elements that are direct children of `#content` will be returned.

2.2 Selecting Specific Siblings

Problem

You need to select only a specific set of siblings of a particular element.

Solution

If you're looking to select the adjacent sibling of a particular element, then you can use the *adjacent sibling* combinator (+). Similar to the child (>) combinator, the sibling combinator expects a selector expression on each side. The righthand expression is the

subject of the selector, and the lefthand expression is the sibling you want to match. Here's some example HTML markup:

```
<div id="content">
    <h1>Main title</h1>
    <h2>Section title</h2>
    <p>Some content...</p>
    <h2>Section title</h2>
    <p>More content...</p>
</div>
```

If you want to select only `<h2>` elements that immediately follow `<h1>` elements, you can use the following selector:

```
jQuery('h1 + h2');
    // Selects ALL H2 elements that are adjacent siblings of H1 elements
```

In this example, only one `<h2>` element will be selected (the first one). The second one is not selected because, while it is a sibling, it is not an *adjacent* sibling of the `<h1>` element.

If, on the other hand, you want to select and filter all siblings of an element, adjacent or not, then you can use jQuery's `siblings()` method to target them, and you can pass an optional selector expression to filter the selection:

```
jQuery('h1').siblings('h2,h3,p');
    // Selects all H2, H3, and P elements that are siblings of H1 elements.
```

Sometimes you'll want to target siblings dependent on their position relative to other elements; for example, here's some typical HTML markup:

```
<ul>
    <li>First item</li>
    <li class="selected">Second item</li>
    <li>Third item</li>
    <li>Fourth item</li>
    <li>Fifth item</li>
</ul>
```

To select all list items beyond the second (after `li.selected`), you could use the following method:

```
jQuery('li.selected').nextAll('li');
```

The `nextAll()` method, just like `siblings()`, accepts a selector expression to filter the selection before it's returned. If you don't pass a selector, then `nextAll()` will return all siblings of the subject element that exist after the subject element, although not before it.

With the preceding example, you could also use another CSS combinator to select all list items beyond the second. The *general sibling* combinator (~) was added in CSS3, so you probably haven't been able to use it in your actual style sheets yet, but fortunately you can use it in jQuery without worrying about support, or lack thereof. It works in exactly the same fashion as the adjacent sibling combinator (+) except that it selects

all siblings that follow, not just the adjacent one. Using the previously specified markup, you would select all list items after `li.selected` with the following selector:

```
jQuery('li.selected ~ li');
```

Discussion

The adjacent sibling combinator can be conceptually tricky to use because it doesn't follow the top-down hierarchical approach of most other selector expressions. Still, it's worth knowing about and is certainly a useful way of selecting what you want with minimal hassle.

The same functionality might be achieved without a selector, in the following way:

```
jQuery('h1').next('h2');
```

The `next()` method can make a nice alternative to the selector syntax, especially in a programmatic setting when you're dealing with jQuery objects as variables, for example:

```
var topHeaders = jQuery('h1');
topHeaders.next('h2').css('margin','0');
```

2.3 Selecting Elements by Index Order

Problem

You need to select elements based on their order among other elements.

Solution

Depending on what you want to do, you have the following filters at your disposal. These may look like CSS pseudoclasses, but in jQuery they're called *filters*:

`:first`
: Matches the first selected element

`:last`
: Matches the last selected element

`:even`
: Matches even elements (zero-indexed)

`:odd`
: Matches odd elements (zero-indexed)

`:eq(n)`
: Matches a single element by its index (*n*)

`:lt(n)`
: Matches all elements with an index below *n*

`:gt(n)`
: Matches all elements with an index above *n*

Assuming the following HTML markup:

```
<ol>
    <li>First item</li>
    <li>Second item</li>
    <li>Third item</li>
    <li>Fourth item</li>
</ol>
```

the first item in the list could be selected in a number of different ways:

```
jQuery('ol li:first');
jQuery('ol li:eq(0)');
jQuery('ol li:lt(1)');
```

Notice that both the `eq()` and `lt()` filters accept a number; since it's zero-indexed, the first item is 0, the second is 1, etc.

A common requirement is to have alternating styles on table rows; this can be achieved with the `:even` and `:odd` filters:

```
<table>
    <tr><td>0</td><td>even</td></tr>
    <tr><td>1</td><td>odd</td></tr>
    <tr><td>2</td><td>even</td></tr>
    <tr><td>3</td><td>odd</td></tr>
    <tr><td>4</td><td>even</td></tr>
</table>
```

You can apply a different class dependent on the index of each table row:

```
jQuery('tr:even').addClass('even');
```

You'd have to specify the corresponding class (`even`) in your CSS style sheet:

```
table tr.even {
    background: #CCC;
}
```

This code would produce the effect shown in Figure 2-1.

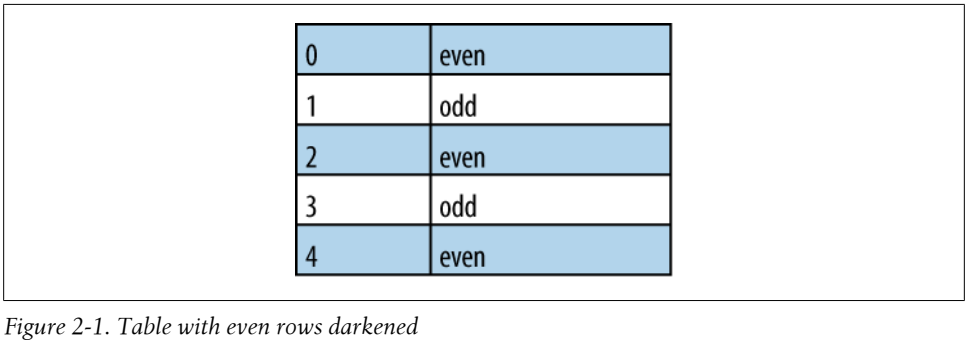
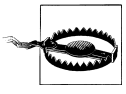


Figure 2-1. Table with even rows darkened

Discussion

As mentioned, an element's index is zero-based, so if an element is the first one, then its index is zero. Apart from that fact, using the preceding filters is very simple. Another thing to note is that these filters require a collection to match against; the index can be determined only if an initial collection is specified. So, this selector wouldn't work:

```
jQuery(':even');
```



Actually, this selector does work, but only because jQuery does some corrective postprocessing of your selector behind the scenes. If no initial collection is specified, then jQuery will assume you meant all elements within the document. So, the selector would actually work, since it's effectively identical to this: `jQuery('*:even')`.

An initial collection is required on the lefthand side of the filter, i.e., something to apply the filter to. The collection can be within an already instantiated jQuery object, as shown here:

```
jQuery('ul li').filter(':first');
```

The filter method is being run on an already instantiated jQuery object (containing the list items).

2.4 Selecting Elements That Are Currently Animating

Problem

You need to select elements based on whether they're animating.

Solution

jQuery offers a convenient filter for this very purpose. The `:animated` filter will match only elements that are currently animating:

```
jQuery('div:animated');
```

This selector would select all `<div>` elements currently animating. Effectively, jQuery is selecting all elements that have a nonempty animation queue.

Discussion

This filter is especially useful when you need to apply a blanket function to all elements that are not currently animated. For example, to begin animating all `<div>` elements that are not already animating, it's as simple as this:

```
jQuery('div:not(div:animated)').animate({height:100});
```

Sometimes you might want to check whether an element is animating. This can be done with jQuery's useful `is()` method:

```
var myElem = jQuery('#elem');
if( myElem.is(':animated') ) {
    // Do something.
}
```

2.5 Selecting Elements Based on What They Contain

Problem

You need to select an element based on what it contains.

Solution

There are normally only two things you would want to query in this respect: the text contents and the element contents (other elements). For the former, you can use the `:contains()` filter:

```
<!-- HTML -->
<span>Hello Bob!</span>

// Select all SPANs with 'Bob' in:
jQuery('span:contains("Bob")');
```

Note that it's case sensitive, so this selector wouldn't match anything if we searched for *bob* (with a lowercase *b*). Also, quotes are not required in all situations, but it's a good practice just in case you encounter a situation where they are required (e.g., when you want to use parentheses).

To test for nested elements, you can use the `:has()` filter. You can pass any valid selector to this filter:

```
jQuery('div:has(p a)');
```

This selector would match all `<div>` elements that encapsulate `<a>` elements (anchors) within `<p>` elements (paragraphs).

Discussion

The `:contains()` filter might not fit your requirements. You may need more control over what text to allow and what to disallow. If you need that control, I suggest using a regular expression and testing against the text of the element, like so:

```
jQuery('p').filter(function(){
    return /(^|\s)(apple|orange|lemon)(\s|$)/.test(jQuery(this).text());
});
```

This would select all paragraphs containing the word *apple*, *orange*, or *lemon*. To read more about jQuery's `filter()` method, have a look at Recipe 2.10.

2.6 Selecting Elements by What They Don't Match

Problem

You need to select a number of elements that don't match a specific selector.

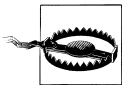
Solution

For this, jQuery gives us the `:not` filter, which you can use in the following way:

```
jQuery('div:not(#content)'); // Select all DIV elements except #content
```

This filter will remove any elements from the current collection that are matched by the passed selector. The selector can be as complex as you like; it doesn't have to be a simple expression, e.g.:

```
jQuery('a:not(div.important a, a.nav)');
// Selects anchors that do not reside within 'div.important' or have the class 'nav'
```



Passing complex selectors to the `:not` filter is possible only in jQuery version 1.3 and beyond. In versions previous to that, only simple selector expressions were acceptable.

Discussion

In addition to the mentioned `:not` filter, jQuery also supplies a method with very similar functionality. This method accepts both selectors and DOM collections/nodes. Here's an example:

```
var $anchors = jQuery('a');
$anchors.click(function(){
    $anchors.not(this).addClass('not-clicked');
});
```

According to this code, when an anchor is clicked, all anchors apart from that one will have the class `not-clicked` added. The `this` keyword refers to the clicked element.

The `not()` method also accepts selectors:

```
$('#nav a').not('a.active');
```

This code selects all anchors residing within `#nav` that do not have a class of `active`.

2.7 Selecting Elements Based on Their Visibility

Problem

You need to select an element based on whether it's visible.

Solution

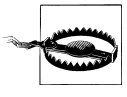
You can use either the `:hidden` or `:visible` filter as necessary:

```
jQuery('div:hidden');
```

Here are some other examples of usage:

```
if (jQuery('#elem').is(':hidden')) {
    // Do something conditionally
}
jQuery('p:visible').hide(); // Hiding only elements that are currently visible
```

Discussion



Since jQuery 1.3.2, these filters have dramatically changed. Before 1.3.2 both filters would respond like you would expect for the CSS `visibility` property, but that is no longer taken into account. Instead, jQuery tests for the height and width of the element in question (relative to its `offsetParent`). If either of these dimensions is zero, then the element is considered hidden; otherwise, it's considered visible.

If you need more control, you can always use jQuery's `filter()` method, which allows you to test the element in any way you want. For example, you may want to select all elements that are set to `display:none` but not those that are set to `visibility:hidden`. Using the `:hidden` filter won't work because it matches elements with either of those characteristics (< v1.3.2) or doesn't take either property into consideration at all (>= v1.3.2):

```
jQuery('*').filter(function(){
    return jQuery(this).css('display') === 'none'
           && jQuery(this).css('visibility') !== 'hidden';
});
```

The preceding code should leave you with a collection of elements that are set to `display:none` but not `visibility:hidden`. Note that, usually, such a selection won't be necessary—the `:hidden` filter is perfectly suitable in most situations.

2.8 Selecting Elements Based on Attributes

Problem

You need to select elements based on attributes and those attributes' values.

Solution

Use an attribute selector to match specific attributes and corresponding values:

```
jQuery('a[href="http://google.com"]');
```

The preceding selector would select all anchor elements with an `href` attribute equal to the value specified (`http://google.com`).

There are a number of ways you can make use of the attribute selector:

`[attr]`
: Matches elements that have the specified attribute

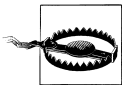
`[attr=val]`
: Matches elements that have the specified attribute with a certain value

`[attr!=val]`
: Matches elements that don't have the specified attribute or value

`[attr^=val]`
: Matches elements with the specified attribute and that start with a certain value

`[attr$=val]`
: Matches elements that have the specified attribute and that end with a certain value

`[attr~=val]`
: Matches elements that contain the specified value with spaces, on either side (i.e., `car` matches `car` but not `cart`)



Prior to jQuery 1.2 you had to use XPath syntax (i.e., putting an `@` sign before an attribute name). This is now deprecated.

You can also combine multiple attribute selectors:

```
// Select all elements with a TITLE and HREF:
jQuery('*[title][href]');
```

Discussion

As always, for special requirements it may be more suitable to use the `filter()` method to more specifically outline what you're looking for:

```
jQuery('a').filter(function(){
    return (new RegExp('http:\/\/(?!' + location.hostname + ')')).test(this.href);
});
```

In this filter, a regular expression is being used to test the `href` attribute of each anchor. It selects all external links within any page.

The attribute selector is especially useful for selecting elements based on slightly varying attributes. For example, if we had the following HTML:

```
<div id="content-sec-1">...</div>
<div id="content-sec-2">...</div>
<div id="content-sec-3">...</div>
<div id="content-sec-4">...</div>
```

we could use the following selector to match all of the `<div>` elements:

```
jQuery('div[id^="content-sec-"]');
```

2.9 Selecting Form Elements by Type

Problem

You need to select form elements based on their types (`hidden`, `text`, `checkbox`, etc.).

Solution

jQuery gives us a bunch of useful filters for this very purpose, as shown in Table 2-1.

Table 2-1. jQuery form filters

jQuery selector syntax	Selects what?
:text	`<input type="text" />`
:password	`<input type="password" />`
:radio	`<input type="radio" />`
:checkbox	`<input type="checkbox" />`
:submit	`<input type="submit" />`
:image	`<input type="image" />`
:reset	`<input type="reset" />`
:button	`<input type="button" />`
:file	`<input type="file" />`
:hidden	`<input type="hidden" />`

So, as an example, if you needed to select all text inputs, you would simply do this: `jQuery(':text');`

There is also an `:input` filter that selects all `input`, `textarea`, `button`, and `select` elements.

Discussion

Note that the `:hidden` filter, as discussed earlier, does not test for the type `hidden`; it works by checking the computed height of the element. This works with input elements of the type `hidden` because they, like other hidden elements, have an `offsetHeight` of zero.

As with all selectors, you can mix and match as desired:

```
jQuery(':input:not(:hidden)');
    // Selects all input elements except those that are hidden.
```

These filters can also be used with regular CSS syntax. For example, selecting all text input elements plus all `<textarea>` elements can be done in the following way:

```
jQuery(':text, textarea');
```

2.10 Selecting an Element with Specific Characteristics

Problem

You need to select an element based not only on its relationship to other elements or simple attribute values but also on varying characteristics such as programmatic states not expressible as selector expressions.

Solution

If you're looking for an element with very specific characteristics, selector expressions may not be the best tool. Using jQuery's DOM filtering method (`filter()`), you can select elements based on anything expressible within a function.

The filter method in jQuery allows you to pass either a string (i.e., a selector expression) or a function. If you pass a function, then its return value will define whether certain elements are selected. The function you pass is run against every element in the current selection; every time the function returns false, the corresponding element is removed from the collection, and every time you return true, the corresponding element is not affected (i.e., it remains in the collection):

```
jQuery('*').filter(function(){
    return !!jQuery(this).css('backgroundImage');
});
```

The preceding code selects all elements with a background image.

The initial collection is of all elements (*); then the `filter()` method is called with a function. This function will return true when a `backgroundImage` is specified for the element in question. The `!!` that you see is a quick way of converting any type in JavaScript to its Boolean expression. Things that evaluate to false include an empty string, the number zero, the value `undefined`, the null type, and, of course, the false Boolean itself. If any of these things are returned from querying the `backgroundImage`, the function will return false, thus removing any elements without background images from the collection. Most of what I just said is not unique to jQuery; it's just JavaScript fundamentals.

In fact, the `!!` is not necessary because jQuery evaluates the return value into a Boolean itself, but keeping it there is still a good idea; anyone looking at your code can be absolutely sure of what you intended (it aids readability).

Within the function you pass to `filter()`, you can refer to the current element via the `this` keyword. To make it into a jQuery object (so you can access and perform jQuery methods), simply wrap it in the jQuery function:

```
this; // Regular element object
jQuery(this); // jQuery object
```

Here are some other filtering examples to spark your imagination:

```
// Select all DIV elements with a width between 100px and 200px:
jQuery('div').filter(function(){
    var width = jQuery(this).width();
    return width > 100 && width < 200;
});

// Select all images with a common image extension:
jQuery('img').filter(function(){
    return /\.(jpe?g|png|bmp|gif)(\?.+)?$/.test(this.src);
});

// Select all elements that have either 10 or 20 children:
jQuery('*').filter(function(){
    var children = jQuery(this).children().length;
    return children === 10 || children === 20;
});
```

Discussion

There will always be several different ways to do something; this is no less true when selecting elements with jQuery. The key differential is usually going to be speed; some ways are fast, others are slow. When you use a complicated selector, you should be thinking about how much processing jQuery has to do in the background. A longer and more complex selector will take longer to return results. jQuery's native methods can sometimes be much faster than using a single selector, plus there's the added benefit of readability. Compare these two techniques:

```
jQuery('div a:not([href^=http://]), p a:not([href^=http://])');

jQuery('div, p').find('a').not('[href^=http://]');
```

The second technique is shorter and much more readable than the first. Testing in Firefox (v3) and Safari (v4) reveals that it's also faster than the first technique.

2.11 Using the Context Parameter

Problem

You've heard of the context parameter but have yet to encounter a situation where it's useful.

Solution

As well as passing a selector expression to `jQuery()` or `$()`, you can pass a second argument that specifies the context. The context is where jQuery will search for the elements matched by your selector expression.

The context parameter is probably one of the most underused of jQuery's features. The way to use it is incredibly simple: pass a selector expression, a jQuery object, a DOM collection, or a DOM node to the context argument, and jQuery will search only for elements within that context.

Here's an example: you want to select all input fields within a form before it's submitted:

```
jQuery('form').bind('submit', function(){
    var allInputs = jQuery('input', this);
    // Now you would do something with 'allInputs'
});
```

Notice that `this` was passed as the second argument; within the handler just shown, `this` refers to the form element. Since it's set as the context, jQuery will only return `input` elements within that form. If we didn't include that second argument, then all of the document's `input` elements would be selected—not what we want.

As mentioned, you can also pass a regular selector as the context:

```
jQuery('p', '#content');
```

The preceding code returns exactly the same collection as the following selector:

```
jQuery('#content p');
```

Specifying a context can aid in readability and speed. It's a useful feature to know about!

Discussion

The default context used by jQuery is `document`, i.e., the topmost item in the DOM hierarchy. Only specify a context if it's different from this default. Using a context can be expressed in the following way:

```
jQuery( context ).find( selector );
```

In fact, this is exactly what jQuery does behind the scenes.

Considering this, if you already have a reference to the context, then you should pass that instead of a selector—there's no point in making jQuery go through the selection process again.

2.12 Creating a Custom Filter Selector

Problem

You need a reusable filter to target specific elements based on their characteristics. You want something that is succinct and can be included within your selector expressions.

Solution

You can extend jQuery's selector expressions under the `jQuery.expr[':']` object; this is an alias for `Sizzle.selectors.filters`. Each new filter expression is defined as a property of this object, like so:

```
jQuery.expr[':'].newFilter = function(elem, index, match){
    return true; // Return true/false like you would on the filter() method
};
```

The function will be run on all elements in the current collection and needs to return true (to keep the element in the collection) or false (to remove the element from the collection). Three bits of information are passed to this function: the element in question, the index of this element among the entire collection, and a match array returned from a regular expression match that contains important information for the more complex expressions.

For example, you might want to target all elements that have a certain property. This filter matches all elements that are displayed inline:

```
jQuery.expr[':'].inline = function(elem) {
    return jQuery(elem).css('display') === 'inline';
};
```

Now that we have created a custom selector, we can use it in any selector expression:

```
// E.g. #1
jQuery('div a:inline').css('color', 'red');
// E.g. #2
jQuery('span').filter(':not(:inline)').css('color', 'blue')
```

jQuery's custom selectors (`:radio`, `:hidden`, etc.) are created in this way.

Discussion

As mentioned, the third parameter passed to your filter function is an array returned from a regular expression match that jQuery performs on the selector string. This match is especially useful if you want to create a filter expression that accepts *parameters*. Let's say that we want to create a selector that queries for data held by jQuery:

```
jQuery('span').data('something', 123);

// We want to be able to do this:
jQuery('*:data(something,123)');
```

The purpose of the selector would be to select all elements that have had data attached to them via jQuery's `data()` method—it specifically targets elements with a datakey of `something`, equal to the number 123.

The proposed filter (`:data`) could be created as follows:

```
jQuery.expr[':'].data = function(elem, index, m) {

    // Remove ":data(" and the trailing ")" from
    // the match, as these parts aren't needed:
    m[0] = m[0].replace(/:data\(|\)$/g, '');

    var regex = new RegExp('([\'"]?)((?:\\\\\\1|.)+?)\\1(,|$)', 'g'),
        // Retrieve data key:
        key = regex.exec( m[0] )[2],
        // Retrieve data value to test against:
        val = regex.exec( m[0] );

    if (val) {
        val = val[2];
    }

    // If a value was passed then we test for it, otherwise
    // we test that the value evaluates to true:
    return val ? jQuery(elem).data(key) == val : !!jQuery(elem).data(key);

};
```

The reason for such a complex regular expression is that we want to make it as flexible as possible. The new selector can be used in a number of different ways:

```
// As we originally mused (above):
jQuery('div:data("something",123)');

// Check if 'something' is a "truthy" value
jQuery('div:data(something)');

// With or without (inner) quotes:
jQuery('div:data(something, "something else")');
```

Now we have a totally new way of querying data held by jQuery on an element.

If you ever want to add more than one new selector at the same time, it's best to use jQuery's `extend()` method:

```
jQuery.extend(jQuery.expr[':'], {
    newFilter1 : function(elem, index, match){
        // Return true or false.
    },
    newFilter2 : function(elem, index, match){
        // Return true or false.
    },
    newFilter3 : function(elem, index, match){
        // Return true or false.
    }
});
```

CHAPTER 3

Beyond the Basics

Ralph Whitbeck

3.0 Introduction

jQuery is a very lightweight library that is capable of helping you do the simple selections of DOM elements on your page. You saw these simple uses in Chapter 1. In this chapter, we'll explore how jQuery can be used to manipulate, traverse, and extend jQuery to infinite possibilities. As lightweight as jQuery is, it was built to be robust and expandable.

3.1 Looping Through a Set of Selected Results

Problem

You need to create a list from your selected set of DOM elements, but performing any action on the selected set is done on the set as a whole. To be able to create a list with each individual element, you'll need to perform a separate action on each element of the selected set.

Solution

Let's say you wanted to make a list of every link within a certain DOM element (perhaps it's a site with a lot of user-provided content, and you wanted to quickly glance at the submitted links being provided by users). We would first create our jQuery selection, `$("div#post a[href]")`, which will select all links with an `href` attribute within the `<div>` with the `id` of `post`. Then we want to loop through each matched element and append it to an array. See the following code example:

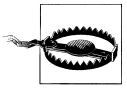
```
var urls = [];
 $("div#post a[href]").each(function(i) {
    urls[i] = $(this).attr('href');
});
```

```
alert(urls.join(","));
```

We were able to make an array because we iterated through each element in the jQuery object by using the `$().each();` method. We are able to access the individual elements and execute jQuery methods against those elements because we wrapped the `this` variable in a jQuery wrapper, `$()`, thus making it a jQuery object.

Discussion

jQuery provides a core method that you can use to loop through your set of selected DOM elements. `$().each()` is jQuery's `for` loop, which will loop through and provide a separate function scope for each element in the set. `$().each();` will iterate exclusively through jQuery objects.



`$().each();` is not the same as the jQuery utility method `jQuery.each(object, callback);`. The `jQuery.each` method is a more generalized iterator method that will iterate through both objects and arrays. See jQuery's online documentation for more information on `jQuery.each()` at *http://docs.jquery.com/Utilities/jQuery.each*.

In each iteration, we are getting the `href` attribute of the current element from the main selection. We are able to get the current DOM element by using the `this` keyword. We then wrap it in the jQuery object, `$(this)`, so that we can perform jQuery methods/actions against it—in our case, pulling the `href` attribute from the DOM element. The last action is to assign the `href` attribute to a global array, `urls`.

Just so we can see what we have, the array URL is joined together with a `,` and displayed to the user in an alert box. We could also have added the list to an unordered list DOM element for display to the user. More practically, we might want to format the list of URLs into JSON format and send it to the server for processing into a database.

Let's look at another example using `$().each();`. This example is probably the most obvious use of `$().each();`. Let's say we have an unordered list of names, and we want each name to stand out. One way to accomplish this is to set an alternate background color for every other list item:

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
    <title>Chapter 3 - Recipe 1 - Looping through a set of selected results</title>
    <style type="text/css">
        .even { background-color: #ffffff; }
        .odd  { background-color: #cccccc; }
    </style>
    <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"
type="text/javascript"></script>
```

```
    <script type="text/javascript">
        (function($){
            $(document).ready(function() {
                $("ul > li").each(function(i) {
                    if (i % 2 == 1)
                    {
                        $(this).addClass("odd");
                    }
                    else
                    {
                        $(this).addClass("even");
                    }
                });
            });
        })(jQuery);
    </script>
</head>
<body>
    <h2>Family Members</h2>
    <ul>
        <li>Ralph</li>
        <li>Hope</li>
        <li>Brandon</li>
        <li>Jordan</li>
        <li>Ralphie</li>
    </ul>
</body>
</html>
```

Figure 3-1 shows the code output.

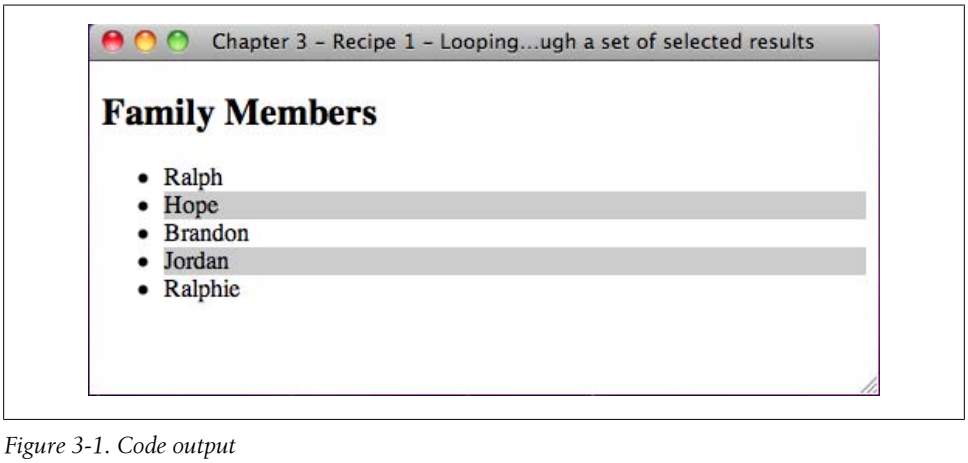


Figure 3-1. Code output

As we iterate through each `<li>` element, we are testing whether the current index, which is passed in as a single argument to the function when executed, modded by 2 is equal to 1. Based on that condition, we either set one CSS class (`.odd`) or another CSS class (`.even`).

Even though this may be the most obvious way to use `$().each()`, it isn't the most efficient way to handle making alternating background colors. We could have accomplished this with one line:

```
$("ul > li:odd").addClass("odd");
```

All we needed to do was set all the `<li>` elements to the class `.even` in the CSS so that we could override the odd `<li>` elements with the `.odd` class with jQuery.

The basic function of `$.each();` is to take the matched set and iterate through each element via reference of the index, perform some action, and iterate to the next element in the matched set until there are no more elements left.

3.2 Reducing the Selection Set to a Specified Item

Problem

A jQuery selector is broad and selects all elements on the page based on your query. The need may rise when you need to select a single item, based on its position, but there isn't an easy way to select that item without editing the code.

Solution

After you make your selection with jQuery, you can chain the `.eq()` method and pass in the index of the selection you want to work with.

The selection index is zero-based, so the first item in the selection would be `$().eq(0);` where 0 represents the first item in the selection. `$().eq(4);` represents the fifth item.

Let's use the end of the season standings for the National Hockey League (NHL) conferences as an example of how we can show which teams made the playoffs and which didn't. What we need to do is list all the teams in each conference in the order they finished the season in. Since the top eight teams in each conference make it to the playoff round, we just need to figure out the eighth entry in each list and draw a line:

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
    <title>Chapter 3 - Recipe 2 - Reducing the selection set to specified item</title>
    <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
```

```
        <script type="text/javascript">
            (function($){
                $(document).ready(function(){
                    $("ol#east > li").eq(7).css("border-bottom", "1px solid #000000");
                    $("ol#west > li").eq(7).css("border-bottom", "1px solid #000000");
                });
            })(jQuery);
        </script>
    </head>
    <body>
        <h2>Eastern Conference</h2>
        <ol id="east">
            <li>Boston Bruins</li>
            <li>Washington Capitals</li>
            <li>New Jersey Devils</li>
            <li>Pittsburgh Penguins</li>
            <li>Philadelphia Flyers</li>
            <li>Carolina Hurricanes</li>
            <li>New York Rangers</li>
            <li>Montreal Canadians</li>
            <li>Florida Panthers</li>
            <li>Buffalo Sabres</li>
            <li>Ottawa Senators</li>
            <li>Toronto Maple Leafs</li>
            <li>Atlanta Thrashers</li>
            <li>Tampa Bay Lightning</li>
            <li>New York Islanders</li>
        </ol>

        <h2>Western Conference</h2>
        <ol id="west">
            <li>San Jose Sharks</li>
            <li>Detroit Red Wings</li>
            <li>Vancouver Canucks</li>
            <li>Chicago Blackhawks</li>
            <li>Calgary Flames</li>
            <li>St. Louis Blues</li>
            <li>Columbus Blue Jackets</li>
            <li>Anaheim Ducks</li>
            <li>Minnesota Wild</li>
            <li>Nashville Predators</li>
            <li>Edmonton Oilers</li>
            <li>Dallas Stars</li>
            <li>Phoenix Coyotes</li>
            <li>Los Angeles Kings</li>
            <li>Colorado Avalanche</li>
        </ol>
    </body>
    </html>
```

Figure 3-2 shows the code output.

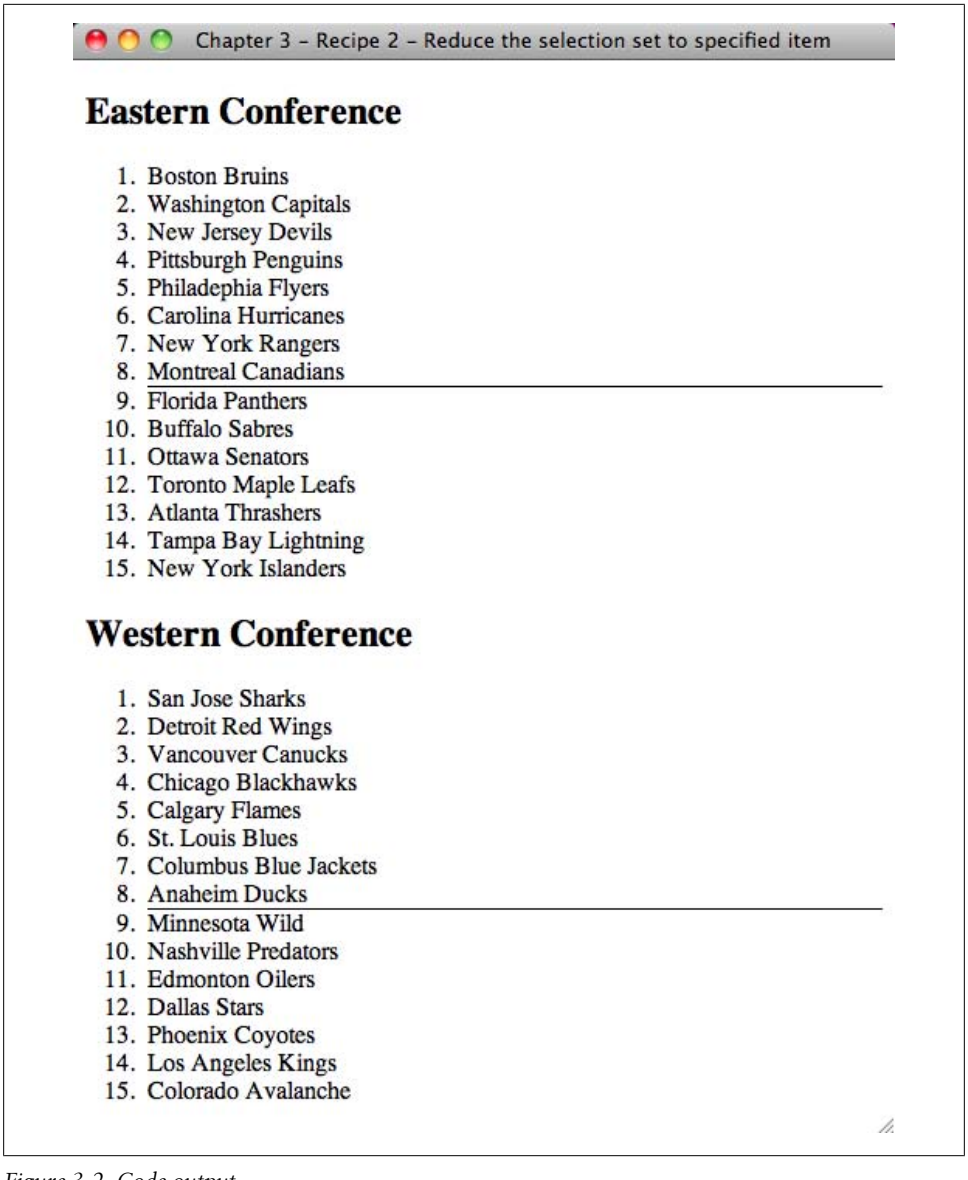


Figure 3-2. Code output

As you can see, we just use an ordered list to list the teams in the order they placed, then we use jQuery to add a bottom border to the eighth item in each list. We need to add an ID to each ordered list so that we can specify each list in a separate query. If we were to do `$("li").eq(7);`, it would select only from the first list because the query would have counted all the `<li>` elements on the page together.

Discussion

The `.eq()` method is used to take a selection set and reduce it to a single item from that set. The argument is the index that you want to reduce your selection to. The index starts at 0 and goes to length –1. If the argument is an invalid index, the method will return an empty set of elements instead of null.

The `.eq()` method is similar to using the `$(":eq()");` right in your selection, but the `.eq()` method allows you to chain to the selection and fine-tune further. For example:

```
$("li").css("background-color","#CCCCCC").eq(0).css("background-color","#ff0000");
```

This will change the background color of all `<li>` elements and then select the first one and give it a different color to signify that it is perhaps a header item.

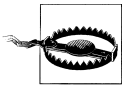
3.3 Convert a Selected jQuery Object into a Raw DOM Object

Problem

Selecting elements on a page with jQuery returns a set as a jQuery object and not as a raw DOM object. Because it's a jQuery object, you can only run jQuery methods against the selected set. To be able to run DOM methods and properties against the selected set, the set needs to be converted to a raw DOM object.

Solution

jQuery provides a core method `get()`, which will convert all matched jQuery objects back into an array of DOM objects. Additionally, you can pass an index value in as an argument of `get()`, which will return the element at the index of the matched set as a DOM object, `$.get(1);`. Now, even though you can get at a single element's DOM object via `$.get(index)`, it is there for historical reasons; the "best practices" way is to use the `[]` notation, `$("div")[1];`.



We are discussing the core `.get()` method, which transforms a jQuery object to a DOM array. We are not discussing the Ajax `get` method, which will load a remote page using an HTTP GET request.

Because `get()` returns an array, you can traverse the array to get at each DOM element. Once it's a DOM element, you can then call traditional DOM properties and methods against it. Let's explore a simple example of pulling the `innerHTML` of an element:

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
```

```
    <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
    <title>Chapter 3 - Recipe 3 - Converting a selected jQuery object into a
raw DOM object</title>
    <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
    <script type="text/javascript">
        (function($){
            $(document).ready(function(){
                var inner = $("div")[0].innerHTML;
                alert(inner);
            });
        })(jQuery);
    </script>
</head>
<body>
    <div>
        <p>
            jQuery, the write less, do more JavaScript library.  Saving the day
for web developers since 2006.
        </p>
    </div>
</body>
</html>
```

Figure 3-3 shows the output.

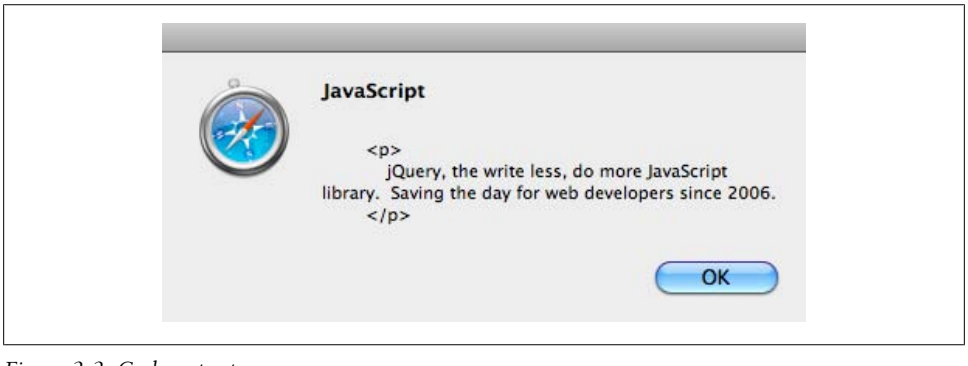


Figure 3-3. Code output

We start by selecting all the `<div>` elements on the page and calling `[0]`. We pass in the index of the selection we want to work with; since there is only one `<div>` on the page, we can pass in index `0`. Finally, we call a property, in this case `innerHTML`, to retrieve the raw DOM element.

Discussion

The core `get()` method can be very useful, as there are some non-JavaScript methods that we can utilize for our advantage. Let's say we have a list and we need to show that list in reverse order. Since `get()` returns an array, we can use native array methods to reverse sort the list and then redisplay the list:

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
    <title>Chapter 3 - Recipe 3 - Converting a selected jQuery object into a raw DOM
object</title>
    <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
    <script type="text/javascript">
    <!--
        (function($){
            $(document).ready(function(){
                var lis = $("ol li").get().reverse();
                $("ol").empty();
                $.each(lis, function(i){
                   $("ol").append("<li>" + lis[i].innerHTML + "</li>");
                });
            });
        })(jQuery);
    //-->
    </script>
</head>
<body>
    <h2>New York Yankees - Batting Line-up</h2>
    <ol>
        <li>Jeter</li>
        <li>Damon</li>
        <li>Teixeira</li>
        <li>Posada</li>
        <li>Swisher</li>
        <li>Cano</li>
        <li>Cabrera</li>
        <li>Molina</li>
        <li>Ransom</li>
    </ol>
</body>
</html>
```

Figure 3-4 shows the output.

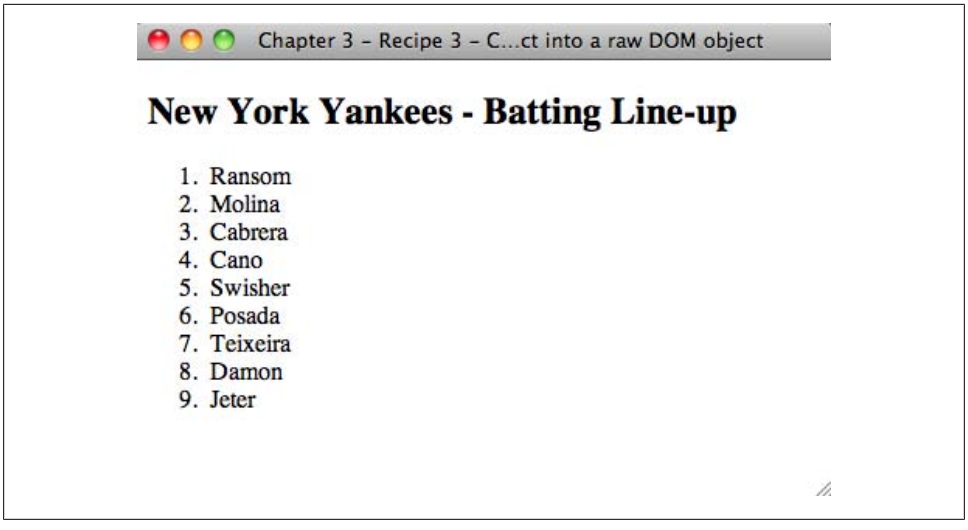


Figure 3-4. Code output

3.4 Getting the Index of an Item in a Selection

Problem

When binding an event for a wide range of selected elements on a page, you need to know exactly which item was clicked from the selected set to "personalize" the action of the bound event.

Solution

When we click an item, we can use the core method `index()` to search through a selection to see what index the item is at:

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
    <title>Chapter 3 - Recipe 4 - Getting the index of an item in a selection</title>
    <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
    <script type="text/javascript">
    <!--
        (function($){
            $(document).ready(function(){
                $("div").click(function() {
                    alert("You clicked on div with an index of " +
$("div").index(this));
```

```
            });
        });
    })(jQuery);
//-->
</script>
</head>
<body>
    <div>click me</div>
    <div class="test">test</div>
    <div>click me</div>
</body>
</html>
```

Figure 3-5 shows the output.

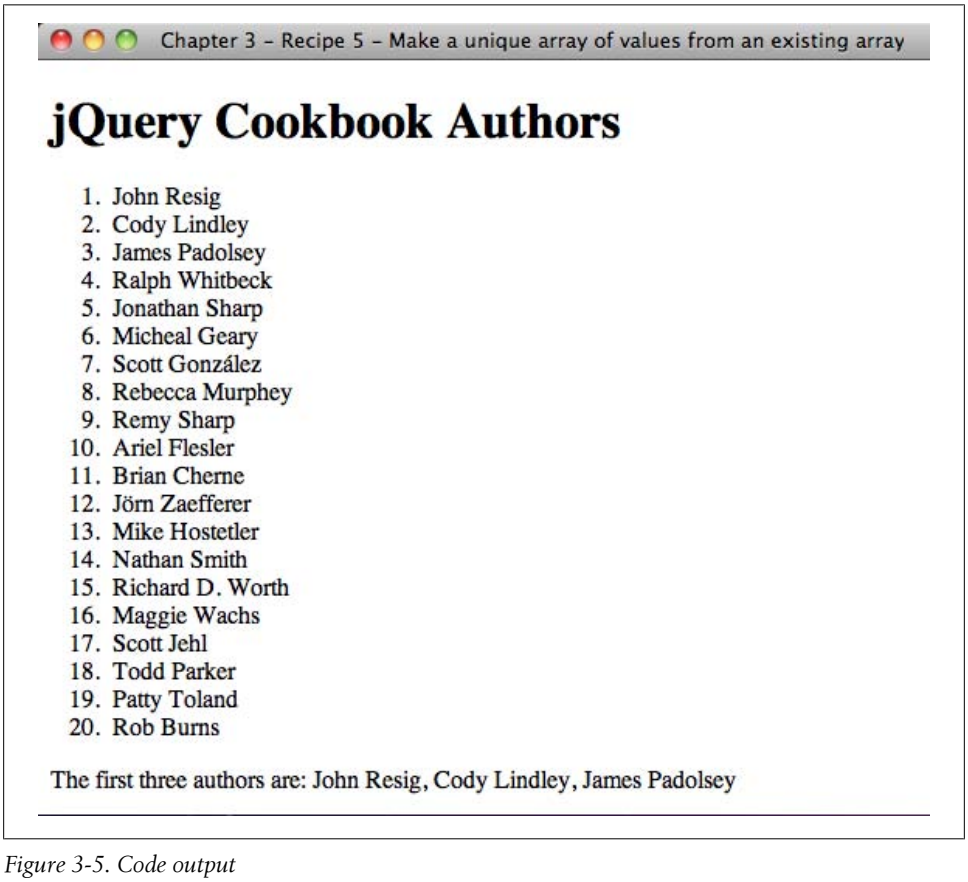


Figure 3-5. Code output

We start by binding all `<div>` elements to a click event. Then when a `<div>` is clicked, we can figure out which `<div>` was clicked by searching for the item in the same selection: `$("div").index(this);`, where `this` is the `<div>` that was clicked.

Discussion

The core method `index()` allows you to get the index of the DOM element you are looking for from a jQuery set. As of jQuery 1.2.6, you can also pass in the index of a jQuery collection to search for. The method will return the index of the first occurrence it finds:

```
var test = $("div.test");

$("div").each(function(i){
    if ($(this).index(test) >= 0)
    {
        //do something
    }
    else
    {
        //do something else
    }
});
```

We'll see whether the `<div>` in the loop matches the collection we saved in the variable `test`, and if so, it will perform a custom action on the matched collection.

If the index method cannot find the subject that was passed in, it will return –1.

3.5 Making a Unique Array of Values from an Existing Array

Problem

You have an ordered list on your page. You select all the `<li>` elements of that list using jQuery; now you need to transform that list into another list.

Solution

Let's say we have a list of people in an ordered list. We would like to display the first three people from that ordered list as a sentence:

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
    <title>Chapter 3 - Recipe 5 - Making a unique array of values from an existing
array</title>
    <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
    <script type="text/javascript">
```

```
    <!--
        (function($){
            $(document).ready(function(){
                var arr = $.map($("LI"), function(item, index){
                    while (index < 3)
                    {
                        return $(item).html();
                    }
                    return null;
                });

                $(document.body).append("<span>The first three authors are: " +
arr.join(", ") + "</span>");
            });
        })(jQuery);
    //-->
    </script>
</head>
<body>
    <h1>jQuery Cookbook Authors</h1>
    <ol>
        <li>John Resig</li>
        <li>Cody Lindley</li>
        <li>James Padolsey</li>
        <li>Ralph Whitbeck</li>
        <li>Jonathan Sharp</li>
        <li>Michael Geary</li>
        <li>Scott González</li>
        <li>Rebecca Murphey</li>
        <li>Remy Sharp</li>
        <li>Ariel Flesler</li>
        <li>Brian Cherne</li>
        <li>Jörn Zaefferer</li>
        <li>Mike Hostetler</li>
        <li>Nathan Smith</li>
        <li>Richard D. Worth</li>
        <li>Maggie Wachs</li>
        <li>Scott Jehl</li>
        <li>Todd Parker</li>
        <li>Patty Toland</li>
        <li>Rob Burns</li>
    </ol>
</body>
</html>
```

Figure 3-6 shows the output.

We start by making an array of the `<li>` elements from the ordered list. We will select all `<li>` elements on the page by using a jQuery selector and pass that in as an argument of the jQuery utility method `$.map()`, which will take an existing array and "map" it into another array. The second argument is the function that will iterate through the array, perform translations, and return a new value to be stored into a new array.

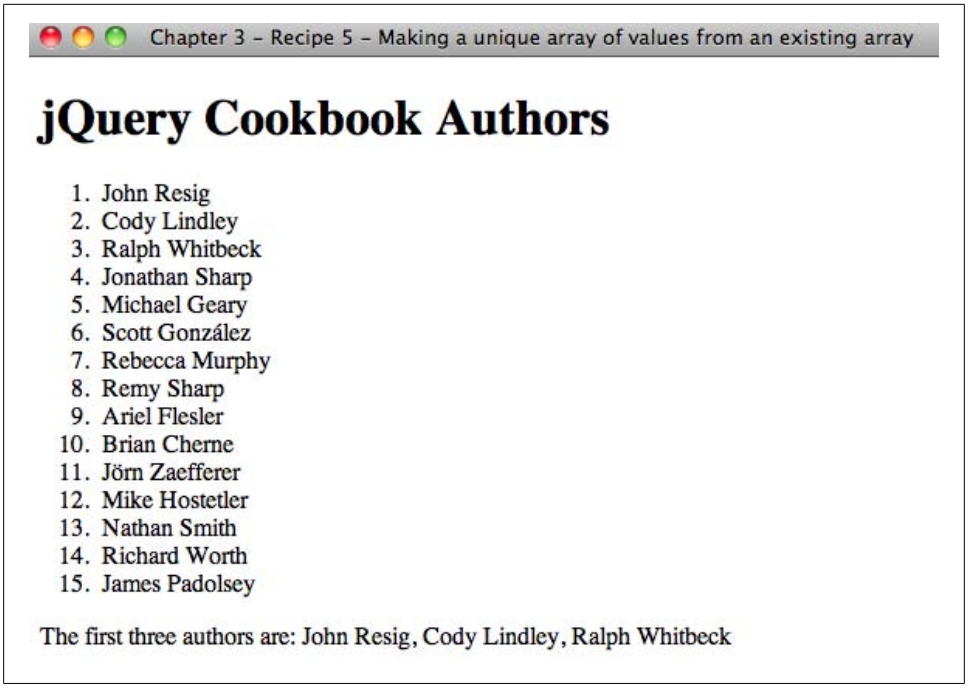


Figure 3-6. Code output

In the preceding example, we iterate through the array we made, return only the `html()` values of the first three list elements, and map these values into a new array. We then take that array and use the join method to make a single string out of the array and inject it into the end of the document.

Discussion

In the solution, we are using the jQuery utility method `$.map()`, which will transform an existing array into another array of items. `$.map()` takes two arguments, an array and a callback function:

```
$.map([1,2,3], function(n,i) { return n+i;});

//Output: [1,3,5]
```

`$.map()` will iterate through each item of the original array and pass in the item to be translated and the index of the current location within the array. The method is expecting a value to be returned. The returned value will be inserted into the new array.

If the null value is returned, no value will be saved into the new array. Returning null basically removes the item from the new array.

3.6 Performing an Action on a Subset of the Selected Set

Problem

You need to perform an action on a set of tags, but there is no way to isolate these tags from all the other tags on the page in a jQuery selection set.

Solution

We can use the `slice()` method to filter the selection set to a subset. We pass it a starting index value and an ending index value, then we can chain our action at the end:

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
    <title>Chapter 3 - Recipe 6 - Performing an action on a subset of the selected
set</title>
    <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
    <script type="text/javascript">
    <!--
        (function($){
            $(document).ready(function(){

  $("p").slice(1,3).wrap("<i></i>");
            });
        })(jQuery);
    //-->
   </script>
</head>
<body>
    <p>
        Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin eget nibh ut
tortor egestas pharetra. Nullam a hendrerit urna. Aenean augue arcu, vestibulum eget
faucibus nec, auctor vel velit. Fusce eget velit non nunc auctor rutrum id et ante.
Donec nec malesuada arcu. Suspendisse eu nibh nulla, congue aliquet metus. Integer
porta dignissim magna, eu facilisis magna luctus ac. Aliquam convallis condimentum
purus, at lacinia nisi semper volutpat. Nulla non risus justo. In ac elit vitae elit
posuere adipiscing.
    </p>
    <p>
        Aliquam gravida metus sit amet orci facilisis eu ultricies risus iaculis. Nunc
tempus tristique magna, molestie adipiscing nibh bibendum vel. Donec sed nisi luctus
sapien scelerisque pretium id eu augue. Mauris ipsum arcu, feugiat non tempor
tincidunt, tincidunt sit amet turpis. Vestibulum scelerisque rutrum luctus. Curabitur
eu ornare nisl. Cras in sem ut eros consequat fringilla nec vitae felis. Nulla
facilisi. Mauris suscipit feugiat odio, a condimentum felis luctus in. Nulla interdum
dictum risus, accumsan dignissim tortor ultricies in. Duis justo mauris, posuere vel
convallis ut, auctor non libero. Ut a diam magna, ut egestas dolor. Nulla convallis,
orci in sodales blandit, lorem augue feugiat nulla, vitae dapibus mi ligula quis
```

```
ligula. Aenean mattis pulvinar est quis bibendum.
    </p>
    <p>
        Donec posuere pulvinar ligula, nec sagittis lacus pharetra ac. Cras nec
tortor mi. Pellentesque et magna vel erat consequat commodo a id nunc. Donec velit
elit, vulputate nec tristique vitae, scelerisque ac sem. Proin blandit quam ut magna
ultrices porttitor. Fusce rhoncus faucibus tincidunt. Cras ac erat lacus, dictum
elementum urna. Nulla facilisi. Praesent ac neque nulla, in rutrum ipsum. Aenean
imperdiet, turpis sit amet porttitor hendrerit, ante dui eleifend purus, eu fermentum
dolor enim et elit.
    </p>
    <p>
        Suspendisse facilisis molestie hendrerit. Aenean congue congue sapien, ac
luctus nulla rutrum vel. Fusce vitae dui urna. Fusce iaculis mattis justo sit amet
varius. Duis velit massa, varius in congue ut, tristique sit amet lorem. Curabitur
porta, mauris non pretium ultrices, justo elit tristique enim, et elementum tellus
enim sit amet felis. Sed sollicitudin rutrum libero sit amet malesuada. Duis vitae
gravida purus. Proin in nunc at ligula bibendum pharetra sit amet sit amet felis.
Integer ut justo at massa ullamcorper sagittis. Mauris blandit tortor lacus,
convallis iaculis libero. Etiam non pellentesque dolor. Fusce ac facilisis ipsum.
Suspendisse eget ornare ligula. Aliquam erat volutpat. Aliquam in porttitor purus.
    </p>
    <p>
        Suspendisse facilisis euismod purus in dictum. Vivamus ac neque ut sapien
fermentum placerat. Sed malesuada pellentesque tempor. Aenean cursus, metus a
lacinia scelerisque, nulla mi malesuada nisi, eget laoreet massa risus eu felis.
Vivamus imperdiet rutrum convallis. Proin porta, nunc a interdum facilisis, nunc dui
aliquet sapien, non consectetur ipsum nisi et felis. Nullam quis ligula nisi, sed
scelerisque arcu. Nam lorem arcu, mollis ac sodales eget, aliquet ac eros. Duis
hendrerit mi vitae odio convallis eget lobortis nibh sodales. Nunc ut nunc vitae
nibh scelerisque tempor at malesuada sapien. Nullam elementum rutrum odio nec aliquet.
    </p>
</body>
</html>
```

Figure 3-7 shows the output.

The preceding example selects the subset starting at index 1 and ending before index 3 and wraps an italics tag around the subselection.

Discussion

The jQuery method `slice()` takes a couple of options; the first is the starting index position, and the second argument, which is optional, is the ending index position. So, say you wanted all `<P>` tags except the first one; you could do `$("p").slice(1)`, and it would start the selection at the second item and select the rest that is in the jQuery selection.

`slice()` also takes a negative number. If a negative number is given, it'll count in from the selection's end. So, `$("p").slice(-1);` will select the last item in the selection. Additionally, `$("p").slice(1, -2);` will start the selection at the second item and select to the second-to-last item.

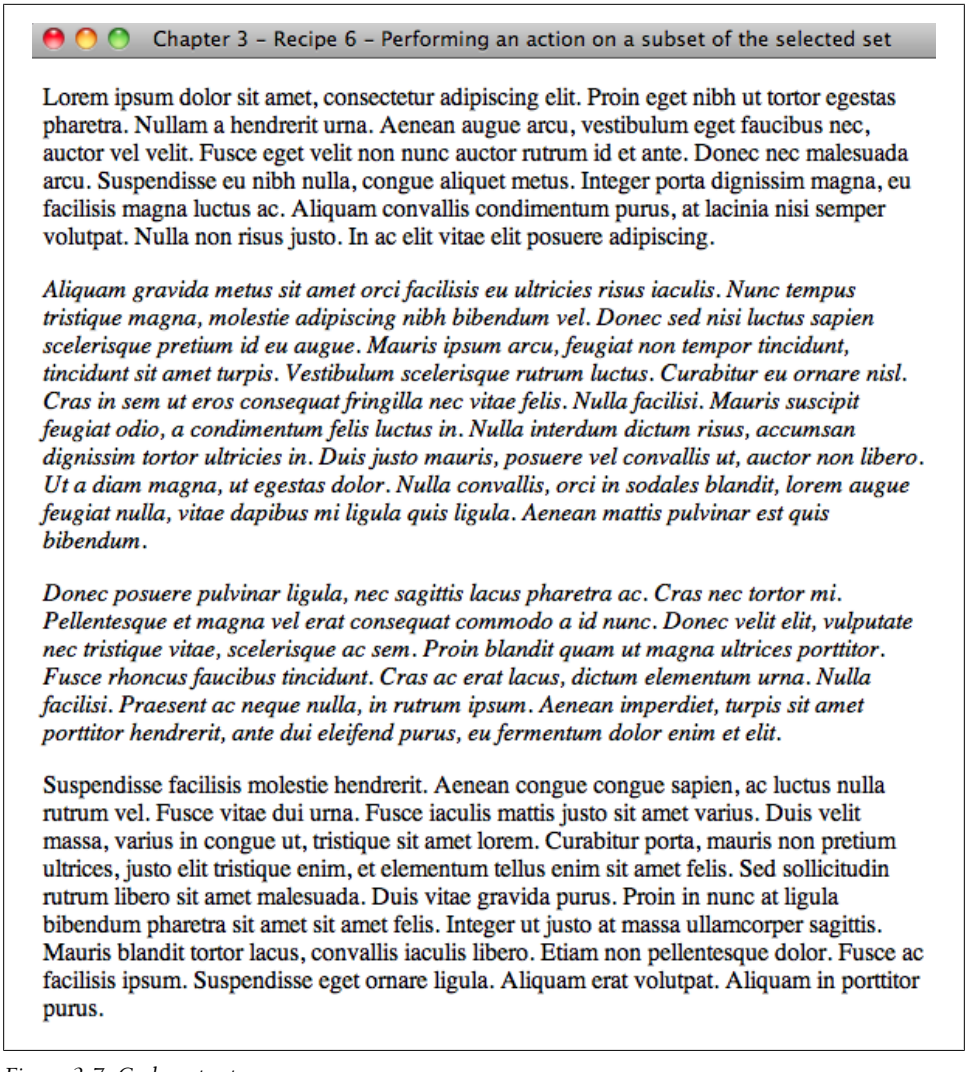


Figure 3-7. Code output

3.7 Configuring jQuery Not to Conflict with Other Libraries

Problem

If jQuery is loaded on the same page as another JavaScript library, both libraries may have implemented the `$` variable, which results in only one of those methods working correctly.

Solution

Let's say you inherit a web page that you need to update, and the previous programmer used another JavaScript library like Prototype, but you still want to use jQuery. This will cause a conflict, and one of the two libraries will not work based on which library is listed last in the page head.

If we just declare both jQuery and Prototype on the same page like so:

```
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/prototype/1.6.0.3/prototype.js"></script>
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
```

this will cause a JavaScript error: *`element.dispatchEvent is not a function in prototype.js`*. Thankfully, jQuery provides a workaround with the `jQuery.noConflict()` method:

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
    <title>Chapter 3 - Recipe 7 - Configuring jQuery to free up a conflict with
another library</title>

    <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/prototype/1.6.0.3/prototype.js"></script>
    <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
    <script type="text/javascript">
    <!--
        jQuery.noConflict();

        // Use jQuery via jQuery(...)
        jQuery(document).ready(function(){
          jQuery("div#jQuery").css("font-weight","bold");
        });

        // Use Prototype with $(...), etc.
        document.observe("dom:loaded", function() {
            $('prototype').setStyle({
               fontSize: '10px'
            });
        });
    //-->
   </script>

</head>
<body>
    <div id="jQuery">Hello, I am a jQuery div</div>
    <div id="prototype">Hello, I am a Prototype div</div>
</body>
</html>
```

Figure 3-8 shows the output.

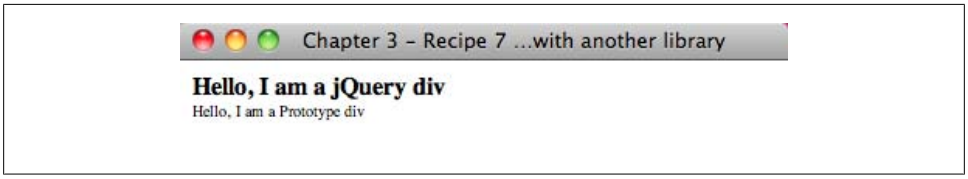


Figure 3-8. Code output

When you call `jQuery.noConflict()`, it gives control of the `$` variable back to whomever implemented it first. Once you free up the `$` variable, you only will be able to access jQuery with the `jQuery` variable. For example, when you used to use `$("div p")`, you would now use `jQuery("div p")`.

Discussion

The jQuery library and virtually all of its plugins are constrained by the jQuery namespace. You shouldn't get a conflict with the jQuery variable and any other library (i.e., Prototype, YUI, etc.). jQuery does however use `$` as a shortcut for the jQuery object. This shortcut definition is what conflicts with other libraries that also use the `$` variable. As we've seen in the solution, we can free jQuery of the `$` shortcut and revert to using the jQuery object.

There is another option. If you want to make sure jQuery won't conflict with another library but still have the benefit of a short name, you can call `jQuery.noConflict()` and assign it to a variable:

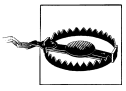
```
var j = jQuery.noConflict();

j(document).ready(function(){
  j("div#jQuery").css("font-weight","bold");
});
```

You can define your own short name by choosing the variable name you assign, `jQuery.noConflict()`.

Finally, another option is to encapsulate your jQuery code inside a closure:

```
jQuery.noConflict();

(function($){
    $("div#jQuery").css("font-weight","bold");
})(jQuery);
```

By using a closure, you temporarily make the `$` variable available to the jQuery object while being run inside the function. Once the function ends, the `$` variable will revert to the library that had initial control.



If you use this technique, you will not be able to use other libraries' methods within the encapsulated function that expect the `$`.

3.8 Adding Functionality with Plugins

Problem

The jQuery library is a small, slick, powerful JavaScript library, but it doesn't come preloaded with every piece of functionality that you may need.

Solution

jQuery was built with extensibility in mind. If the core jQuery library can't do what you want, chances are a jQuery plugin author has written a plugin that will handle your need, probably in as little as one line of code.

To include a plugin on your page, all you need to do is download the plugin *.js* file, include the jQuery library on the page, then immediately after, include your plugin on the page. Then, in either another *.js* file or in a script block on the page, you'll typically need to call the plugin and provide any options that may be required.

Here is an example using the jQuery cycle plugin developed by Mike Alsup:

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
    <title>Chapter 3 - Recipe 8 - Adding Functionality with Plugins</title>
    <style type="text/css">
        .pics {
            height:  232px;
            width:   232px;
            padding: 0;
            margin:  0;
        }

        .pics img {
            padding: 15px;
            border:  1px solid #ccc;
            background-color: #eee;
            width:  200px;
            height: 200px;
            top:  0;
            left: 0
        }
    </style>
    <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
    <!--Now include your plugin declarations after you've declared jQuery on the page-->
```

```
    <script type="text/javascript" src="scripts/2.8/jquery.cycle.all.min.js?
v2.60"></script>
    <script type="text/javascript">
    <!--
        (function($){
            $(document).ready(function(){
                $('.pics').cycle('fade');
            });
        })(jQuery);
    //-->
    </script>

</head>
<body>
    <div class="pics">
        <img src="images/2.8/beach1.jpg" width="200" height="200" alt="Beach 1" />
        <img src="images/2.8/beach2.jpg" width="200" height="200" alt="Beach 2" />
        <img src="images/2.8/beach3.jpg" width="200" height="200" alt="Beach 3" />
    </div>
</body>
</html>
```

Figure 3-9 shows the output.

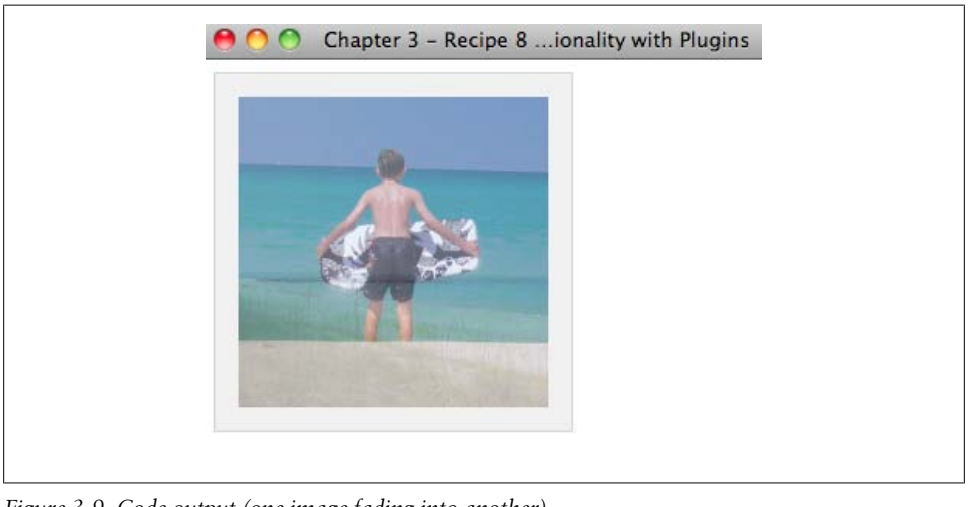


Figure 3-9. Code output (one image fading into another)

With one line of code, we are able to make a slideshow effect that will show one image at a time and then fade to the next image automatically. The cycle plugin is also extensible because it was written so developers can provide different options to have different transition effects and layouts.

Discussion

jQuery has one of the largest communities of developers of any of the JavaScript libraries. This large community contributes to a large base of plugins and tutorials that are

available on the Web. jQuery hosts a repository of plugins that have been written and submitted to *http://plugins.jquery.com* by the authors. There are currently more than 1,600 plugins listed in the repository, and you can find plugins in many different categories. Plugin authors are invited to submit their plugins and to give a description, a link to the plugin, and a link to the plugin's documentation. The repository makes it easy for developers to search for the specific functionality they want.

Chances are that, as a developer, you will eventually find a plugin that meets your requirements. But on the off chance that a plugin doesn't exist, creating a plugin yourself is fairly straightforward. Here are some points to remember:

- Name your file *jquery.[name of plugin].js*, as in *jquery.debug.js*.
- All new methods are attached to the `jQuery.fn` object; all functions to the jQuery object.
- Inside methods, `this` is a reference to the current jQuery object.
- Any methods or functions you attach must have a semicolon (`;`) at the end—otherwise, the code will break when compressed.
- Your method must return the jQuery object, unless explicitly noted otherwise.
- You should use `this.each` to iterate over the current set of matched elements—it produces clean and compatible code that way.
- Always use `jQuery` instead of `$` inside your plugin code—that allows users to change the alias for jQuery in a single place.

For more information and examples on creating plugins, you can go to the Authoring page on the jQuery documentation site, or you can skip ahead to Chapter 12 where Mike Hostetler will go into more detail.

3.9 Determining the Exact Query That Was Used

Problem

While writing a plugin or a method that extends jQuery, you need to know exactly what the selection and the context used when calling the method so that the method can be recalled.

Solution

We can use the core properties `.selector` and `.context` in conjunction with each other so we can re-create the original query that was passed through. We need to use both in conjunction because not all queries to our function or plugin will be within the default document context:

```
<!DOCTYPE html
     PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
    <title>Chapter 3 - Recipe 9 - Determining the exact query that was used</title>
    <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
    <script type="text/javascript">
    <!--
        (function($){
            $.fn.ShowQuery = function(i) {
                alert("$(\""+ $(this).selector + "\", " + $(this).context +")");
                if (i < 3)
                {
                    $($(this).selector, $(this).context).ShowQuery(i+1);
                }
            };
            $("div").ShowQuery(1);
        })(jQuery);
    //-->
    </script>
</head>
<body>
    <div>
        This is a div.
    </div>
</body>
</html>
```

Figure 3-10 shows the output.

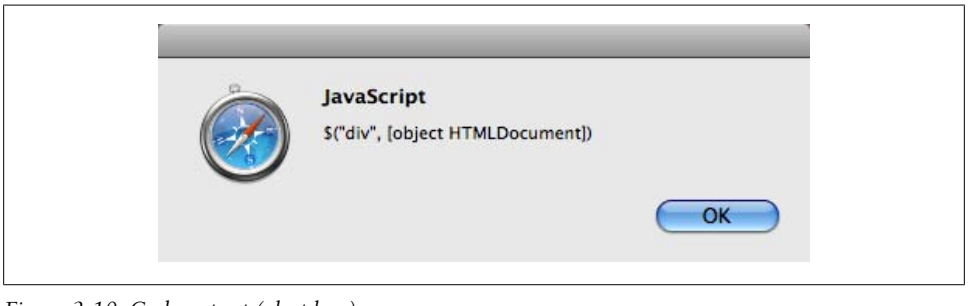


Figure 3-10. Code output (alert box)

Discussion

In the preceding example, we define a method that can be called from a jQuery selection, `ShowQuery`. Within that method, we alert the query as it was passed in and then recursively recall `ShowQuery` again with the same jQuery selector. The `if` statement is there so that we don't get into a recursive loop.

The core properties `.selector` and `.context` were introduced in jQuery 1.3, which was released in January 2009. These methods are geared more toward plugin developers who may need to perform an action against the original query passed in. A potential use case of using these methods is to rerun the selection query or to check to see whether an element is in the selection.

`.selector` returns as a string the actual selector that was used to match the given elements. `.selector` will return the whole selector if, say, the selection is broken up where there is a selector and then the matched set is narrowed with the use of the `find()` method:

```
$("div").find("a").selector;

//returns: "div a"
```

`.context` will return the DOM node originally passed in to `jQuery()`. If no context was set in the selector, the context will default to the document.

CHAPTER 4

jQuery Utilities

Jonathan Sharp

4.0 Introduction

Often, when thinking and talking about jQuery, the main concepts that come to mind are DOM and style manipulation and behavior (events). Yet there are also a number of "core" features and utility functions tucked away for the developer's benefit. This chapter is focused on exposing, disclosing, and explaining these not-so-common utility methods of jQuery.

4.1 Detecting Features with jQuery.support

Problem

You need to attach a special `click` handler to all anchor tags that have just a hash for the current page, and you don't want to risk it breaking because of browser support issues.

Solution

```
(function($) {
    $(document).ready(function() {
     $('a')
         .filter(function() {
             var href = $(this).attr('href');
             // Normalize the URL
             if ( !jQuery.support.hrefNormalized ) {
                 var loc = window.location;
                 href = href.replace( loc.protocol + '//' + loc.host + loc.pathname,
'');
             }
             // This anchor tag is of the form <a href="#hash">
             return ( href.substr(0, 1) == '#' );
        })
```

```
            .click(function() {
                // Special click handler code
            });
        });
    })(jQuery);
```

Discussion

The `jQuery.support` object was added in version 1.3 and contains Boolean flags to help write code using browser feature detection. In our example, Internet Explorer (IE) has a different behavior in how it handles the `href` attribute. IE will return the full URL instead of the exact `href` attribute. Using the `hrefNormalized` attribute, we have future-proofed our solution in the event that a later version of IE changes this behavior. Otherwise, we would have needed a conditional that contained specific browser versions. While it may be tempting, it is best to avoid this approach because it requires future maintenance as new versions of browsers are released. Another reason to avoid targeting specific browsers is that it is possible for clients to intentionally or unintentionally report an incorrect user agent string. In addition to the `hrefNormalized` attribute, a number of additional attributes exist:

`boxModel`
: True if the browser renders according to the W3C CSS box model specification

`cssFloat`
: True if `style.cssFloat` is used to get the current CSS float value

`hrefNormalized`
: True if the browser leaves intact the results from `getAttribute('href')`

`htmlSerialize`
: True if the browser properly serializes link elements with the `innerHTML` attribute

`leadingWhitespace`
: True if the browser preserves leading whitespace when `innerHTML` is used

`noCloneEvent`
: True if the browser does not clone event handlers when elements are cloned

`objectAll`
: True if `getElementsByTagName('*')` on an element returns all descendant elements

`opacity`
: True if the browser can interpret the CSS opacity style

`scriptEval`
: True if using `appendChild` for a `<script>` tag will execute the script

`style`
: True if `getAttribute('style')` is able to return the inline style specified by an element

`tbody`
: True if the browser allows `<table>` elements without a `<tbody>` element

4.2 Iterating Over Arrays and Objects with jQuery.each

Problem

You need to iterate or loop over each element in an array or attribute of an object.

Solution

```
(function($) {
    $(document).ready(function() {
        var months = [  'January', 'February', 'March', 'April', 'May',
                        'June', 'July', 'August', 'September', 'October',
                        'November', 'December'];
        $.each(months, function(index, value) {
            $('#months').append('<li>' + value + '</li>');
        });

        var days = {    Sunday: 0, Monday: 1, Tuesday: 2, Wednesday: 3,
                        Thursday: 4, Friday: 5, Saturday: 6 };
        $.each(days, function(key, value) {
            $('#days').append('<li>' + key + ' (' + value + ')</li>');
        });
    });
})(jQuery);
```

Discussion

In this recipe, we iterate over both an array and an object using `$.each()`, which provides an elegant interface to the common task of iteration. The first argument to the `$.each()` method is the array or object to iterate over, with the second argument being the callback method that is executed for each element. (Note that this is slightly different from the jQuery collection method `$('div').each()`, whose first argument is the callback function.)

When the callback function defined by the developer is executed, the `this` variable is set to the value of the element currently being iterated. Thus, the previous recipe could be rewritten as follows:

```
(function($) {
    $(document).ready(function() {
        var months = [  'January', 'February', 'March', 'April', 'May',
                        'June', 'July', 'August', 'September', 'October',
                        'November', 'December'];
        $.each(months, function() {
            $('#months').append('<li>' + this + '</li>');
        });

        var days = {    Sunday: 0, Monday: 1, Tuesday: 2, Wednesday: 3,
                        Thursday: 4, Friday: 5, Saturday: 6 };
        $.each(days, function(key) {
            $('#days').append('<li>' + key + ' (' + this + ')</li>');
        });
```

```
    });
})(jQuery);
```

4.3 Filtering Arrays with jQuery.grep

Problem

You need to filter and remove elements in an array.

Solution

```
(function($) {
    $(document).ready(function() {
        var months = [  'January', 'February', 'March', 'April', 'May',
                        'June', 'July', 'August', 'September', 'October',
                        'November', 'December'];
        months = $.grep(months, function(value, i) {
            return ( value.indexOf('J') == 0 );
        });
        $('#months').html( '<li>' + months.join('</li><li>') + '</li>' );
    });
})(jQuery);
```

Discussion

This recipe uses the `$.grep()` method to filter the `months` array so that it only includes entries that begin with the capital letter `J`. The `$.grep` method returns the filtered array. The callback method defined by the developer takes two arguments and is expected to return a Boolean value of `true` to keep an element or `false` to have it removed. The first argument specified is the value of the array element (in this case, the month), and the second argument passed in is the incremental value of the number of times the `$.grep()` method has looped. So, for example, if you want to remove every other month, you could test whether `( i % 2 ) == 0`, which returns the remainder of i / 2. (The `%` is the modulus operator, which returns the remainder of a division operation. So, when `i = 4`, `i` divided by `2` has a remainder of `0`.)

```
(function($) {
    $(document).ready(function() {
        var months = [  'January', 'February', 'March', 'April', 'May',
                        'June', 'July', 'August', 'September', 'October',
                        'November', 'December'];
        months = $.grep(months, function(value, i) {
            return ( i % 2 ) == 0;
        });
        $('#months').html( '<li>' + months.join('</li><li>') + '</li>' );
    });
})(jQuery);
```

4.4 Iterating and Modifying Array Entries with jQuery.map

Problem

You need to loop over each element in an array and modify its value.

Solution

```
(function($) {
    $(document).ready(function() {
        var months = [  'January', 'February', 'March', 'April', 'May',
                        'June', 'July', 'August', 'September', 'October',
                        'November', 'December'];
        months = $.map(months, function(value, i) {
            return value.substr(0, 3);
        });
        $('#months').html( '<li>' + months.join('</li><li>') + '</li>' );
    });
})(jQuery);
```

Discussion

In this recipe, `$.map()` is iterating over the `months` array and returns the abbreviation (first three characters). The `$.map()` method takes an array and a callback method as arguments and iterates over each array element executing the callback as defined by the developer. The array entry will be updated with the return value of the callback.

4.5 Combining Two Arrays with jQuery.merge

Problem

You have two arrays that you need to combine or concatenate.

Solution

```
(function($) {
    $(document).ready(function() {
        var horseBreeds = ['Quarter Horse', 'Thoroughbred', 'Arabian'];
        var draftBreeds = ['Belgian', 'Percheron'];

        var breeds = $.merge( horseBreeds, draftBreeds );
        $('#horses').html( '<li>' + breeds.join('</li><li>') + '</li>' );
    });
})(jQuery);
```

Discussion

In this example, we have two arrays that contain a list of horse breeds. The arrays are combined in the order of first + second. So, the final `breeds` array will look like this:

```
['Quarter Horse', 'Thoroughbred', 'Arabian', 'Belgian', 'Percheron']
```

4.6 Filtering Out Duplicate Array Entries with jQuery.unique

Problem

You have two jQuery DOM collections that need to have duplicate elements removed:

```
(function($) {
    $(document).ready(function() {
        var animals = $('li.animals').get();
        var horses = $('li.horses').get();
        $('#animals')
            .append( $(animals).clone() )
            .append( $(horses).clone() );
    });
})(jQuery);
```

Solution

```
(function($) {
    $(document).ready(function() {
        var animals = $('li.animals').get();
        var horses = $('li.horses').get();
        var tmp = $.merge( animals, horses );
        tmp = $.unique( tmp );
        $('#animals').append( $(tmp).clone() );
    });
})(jQuery);
```

Discussion

jQuery's `$.unique()` function will remove duplicate DOM elements from an array or collection. In the previous recipe, we combine the `animals` and `horses` arrays using `$.merge()`. jQuery makes use of `$.unique()` throughout most of its core and internal functions such as `.find()` and `.add()`. Thus, the most common use case for this method is when operating on an array of elements not constructed with jQuery.

4.7 Testing Callback Functions with jQuery.isFunction

Problem

You have written a plugin and need to test whether one of the settings is a valid callback function.

Solution

```
(function($) {
    $.fn.myPlugin = function(settings) {
        return this.each(function() {
            settings = $.extend({ onShow: null }, settings);
            $(this).show();
            if ( $.isFunction( settings.onShow ) ) {
                settings.onShow.call(this);
            }
        });
    };
    $(document).ready(function() {
        $('div').myPlugin({
            onShow: function() {
                alert('My callback!');
            }
        });
    });
})(jQuery);
```

Discussion

While the JavaScript language provides the `typeof` operator, inconsistent results and edge cases across web browsers need to be taken into account. jQuery provides the `.isFunction()` method to ease the developer's job. Worth pointing out is that since version 1.3, this method works for user-defined functions and returns inconsistent results with built-in language functions such as this:

```
jQuery.isFunction( document.getElementById );
```

which returns false in versions of Internet Explorer.

4.8 Removing Whitespace from Strings or Form Values with jQuery.trim

Problem

You have an input form and need to remove the whitespace that a user may have entered at either the beginning or end of a string.

Solution

```
<input type="text" name="first_name" class="cleanup" />
<input type="text" name="last_name" class="cleanup" />

(function($) {
    $(document).ready(function() {
        $('input.cleanup').blur(function() {
            var value = $.trim( $(this).val() );
            $(this).val( value );
```

```
        });
    });
})(jQuery);
```

Discussion

Upon the user blurring a field, the value as entered by the user—`$(this).val()`—is retrieved and passed through the `$.trim()` method that strips all whitespace characters (space, tab, and newline characters) from the beginning and end of the string. The trimmed string is then set as the value of the input field again.

4.9 Attaching Objects and Data to DOM with jQuery.data

Problem

Given the following DOM code:

```
var node = document.getElementById('myId');
node.onclick = function() {
    // Click handler
};
node.myObject = {
    label: document.getElementById('myLabel')
};
```

you have metadata associated with a DOM element for easy reference. Because of flawed garbage collection implementations of some web browsers, the preceding code can cause memory leaks.

Solution

Properties added to an object or DOM node at runtime (called *expandos*) exhibit a number of issues because of flawed garbage collection implementations in some web browsers. jQuery provides developers with an intuitive and elegant method called `.data()` that aids developers in avoiding memory leak issues altogether:

```
$('#myId').data('myObject', {
    label: $('#myLabel')[0]
});

var myObject = $('#myId').data('myObject');
myObject.label;
```

Discussion

In this recipe, we use the `.data()` method, which manages access to our data and provides a clean separation of data and markup.

One of the other benefits of using the `data()` method is that it implicitly triggers `getData` and `setData` events on the target element. So, given the following HTML:

```
<div id="time" class="updateTime"></div>
```

we can separate our concerns (model and view) by attaching a handler for the `setData` event, which receives three arguments (the `event` object, data `key`, and data `value`):

```
// Listen for new data
$(document).bind('setData', function(evt, key, value) {
    if ( key == 'clock' ) {
        $('.updateTime').html( value );
    }
});
```

The `setData` event is then triggered every time we call `.data()` on the document element:

```
// Update the 'time' data on any element with the class 'updateTime'
setInterval(function() {
    $(document).data('clock', (new Date()).toString() );
}, 1000);
```

So, in the previous recipe, every 1 second (1,000 milliseconds) we update the `clock` data property on the `document` object, which triggers the `setData` event bound to the `document`, which in turn updates our display of the current time.

4.10 Extending Objects with jQuery.extend

Problem

You have developed a plugin and need to provide default options allowing end users to overwrite them.

Solution

```
(function($) {
   $.fn.myPlugin = function(options) {
        options = $.extend({
            message: 'Hello world',
            css: {
                color: 'red'
            }
        }, options);
        return this.each(function() {
            $(this).css(options.css).html(options.message);
        });
    };
})(jQuery);
```

Discussion

In this recipe, we use the `$.extend()` method provided by jQuery. `$.extend()` will return a reference to the first object passed in with the latter objects overwriting any properties they define. The following code demonstrates how this works in practice:

```
var obj = { hello: 'world' };
obj = $.extend(obj, { hello: 'big world' }, { foo: 'bar' });

alert( obj.hello ); // Alerts 'big world'
alert( obj.foo ); // Alerts 'bar';
```

This allows for `myPlugin()` in our recipe to accept an `options` object that will overwrite our default settings. The following code shows how an end user would overwrite the default CSS `color` setting:

```
$('div').myPlugin({ css: { color: 'blue' } });
```

One special case of the `$.extend()` method is that when given a single object, it will extend the base jQuery object. Thus, we could define our plugin as follows to extend the jQuery core:

```
$.fn.extend({
    myPlugin: function() {
        options = $.extend({
            message: 'Hello world',
            css: {
                color: 'red'
            }
        }, options);
        return this.each(function() {
            $(this).css(options.css).html(options.message);
        });
    }
});
```

`$.extend()` also provides a facility for a deep (or recursive) copy. This is accomplished by passing in Boolean `true` as the first parameter. Here is an example of how a deep copy would work:

```
var obj1 = { foo: { bar: '123', baz: '456' }, hello: 'world' };
var obj2 = { foo: { car: '789' } };

var obj3 = $.extend( obj1, obj2 );
```

Without passing in `true`, `obj3` would be as follows:

```
{ foo: { car: '789 }, hello: 'world' }
```

If we specify a deep copy, `obj3` would be as follows after recursively copying all properties:

```
var obj3 = $.extend( true, obj1, obj2 );
// obj3
{ foo: { bar: '123', baz: '456', car: '789 }, hello: 'world' }
```

CHAPTER 5

Faster, Simpler, More Fun

Michael Geary and Scott González

5.0 Introduction

Nearly every day, someone asks on the jQuery Google Group how they can make their code simpler or faster, or how to debug a piece of code that isn't working.

This chapter will help you simplify your jQuery code, making it easier to read and more fun to work on. And we'll share some tips for finding and fixing those bugs.

We'll also help you make your code run faster, and equally important, find out which parts of your code you need to speed up. So your site's visitors will have more fun using the snappy pages on your site.

That's what we call a win-win situation. Happy coding!

5.1 That's Not jQuery, It's JavaScript!

Problem

You're a web designer who is new to jQuery, and you're having trouble with the syntax of an `if/else` statement. You know it must be a simple problem, and before asking on the jQuery mailing list, you do your homework: you search the jQuery documentation and find nothing. Web searches for terms like *jquery if else statement* aren't proving helpful either.

You also need to split an email address into two parts, separating it at the `@` sign. You've heard that there is a function to split strings, but there doesn't seem to be any information in the jQuery documentation about this either.

Is jQuery really that poorly documented?

Solution

The `if/else` statement and the `.split()` method for strings are part of JavaScript, not part of jQuery.

So, these web searches will turn up more useful results:

javascript if else statement
javascript split string

Discussion

JavaScript experts, please don't bite the newbies.

Newbies, don't feel bad if you've scratched your head over something like this.

If you're an old pro at JavaScript, you may laugh at these questions. But they come up fairly often on the jQuery mailing list, and understandably so. jQuery is designed to make simple JavaScript coding so easy that someone who's never programmed before can pick up the basics and add useful effects to a page, without having to learn a "real" programming language.

But jQuery *is* JavaScript. jQuery itself is 100% pure JavaScript code, and every line of jQuery you write is also a line of JavaScript.

You can indeed get many simple tasks done with jQuery without really understanding its relationship to JavaScript, but the more you learn about the underlying language, the more productive—and less frustrating—your jQuery experience will be.

5.2 What's Wrong with $(this)?

Problem

You have an event handler that adds a class to a DOM element, waits one second using `setTimeout()`, and then removes that class:

```
$(document).ready( function() {
    $('.clicky').click( function() {
        $(this).addClass('clicked');
        setTimeout( function() {
            $(this).removeClass('clicked');
        }, 1000 );
     });
});
```

The class gets added when you click, but it never gets removed. You have confirmed that the code inside `setTimeout()` is being called, but it doesn't seem to do anything. You've used `.removeClass()` before, and that code looks correct. You are using `$(this)` the same way in both places, but it doesn't seem to work inside the `setTimeout()` call.

Solution

Save `this` in a variable before calling `setTimeout()`:

```
$(document).ready( function() {
    $('.clicky').click( function() {
        var element = this;
        $(element).addClass('clicked');
        setTimeout( function() {
            $(element).removeClass('clicked');
        }, 1000 );
     });
});
```

Even better, since you're calling `$()` in both places, follow the advice in Recipe 5.3 and copy `$(this)` to a variable instead of `this`:

```
$(document).ready( function() {
    $('.clicky').click( function() {
        var $element = $(this);
        $element.addClass('clicked');
        setTimeout( function() {
            $element.removeClass('clicked');
        }, 1000 );
     });
});
```

Discussion

What *is* `$(this)` anyway, and why doesn't it always work? It's easier to understand if you separate it into its two parts, `$()` and `this`.

`$()` looks mysterious, but it really isn't: it's just a function call. `$` is a reference to the `jQuery` function, so `$()` is simply a shorter way to write `jQuery()`. It's just an ordinary JavaScript function call that happens to return an object.

If you're using another JavaScript library that redefines `$`, that's a different matter—but then you wouldn't use `$()` in your jQuery code; you'd use `jQuery()` or a custom alias.

`this` is one of the more confusing features in JavaScript, because it's used for so many different things. In object-oriented JavaScript programming, `this` is used in an object's methods to refer to that object, just like `self` in Python or Ruby:

```
function Foo( value ) {
    this.value = value;
}

Foo.prototype.alert = function() {
    alert( this.value );
};
```

```
var foo = new Foo( 'bar' );
foo.alert();  // 'bar'
```

In the code for a traditional *onevent* attribute, `this` refers to the element receiving the event—but only in the attribute itself, not in a function called from the attribute:

```
<a href="#" id="test" onclick="clicked(this);">Test</a>

function clicked( it ) {
    alert( it.id );            // 'test'
    alert( this.id );          // undefined
    alert( this === window );  // true (what?)
}
```

As you can see from the third `alert()`, `this` is actually the `window` object inside the function. For historical reasons, `window` is the "default" meaning of `this` when a function is called directly (i.e., not called as a method of an object).

In a jQuery event handler, `this` is the DOM element handling the event, so `$(this)` is a jQuery wrapper for that DOM element. That's why `$(this).addClass()` works as expected in our "Problem" code.

But the code then calls `setTimeout()`, and `setTimeout()` works like a direct function call: `this` is the `window` object. So when the code calls `$(this).removeClass()`, it's actually trying to remove the class from the `window` object!

Why does copying `this` or `$(this)` into a local variable fix this? (Pun intended.) JavaScript creates a *closure* for the parameters and local variables of a function.

Closures may seem mysterious at first, but they really boil down to three simple rules:

- You can nest JavaScript functions one inside another, with multiple levels of nesting.
- A function can read and write not only its own parameters and local variables but also those of any functions it's nested in.
- The previous rule *always* works, even if the outer function has already returned and the inner function is called later (e.g., an event handler or `setTimeout()` callback).

These rules apply equally to all functions, both named and anonymous. However, `this` is not a function parameter or local variable—it's a special JavaScript keyword—so these rules do not apply. By copying the value of `this` into a local variable, we take advantage of the closure to make that value available in any nested functions.

5.3 Removing Redundant Repetition

Problem

You need to hide, show, or otherwise manipulate some DOM elements when the page loads, and you also need to take the same actions later in response to a couple of different events:

```
$(document).ready( function() {

    // Set visibility at startup
    $('#state').toggle( $('#country').val() == 'US' );
    $('#province').toggle( $('#country').val() == 'CA' );

    // Update visibility when country selector changes via mouse
    $('#country').change( function() {
        $('#state').toggle( $(this).val() == 'US' );
        $('#province').toggle( $(this).val() == 'CA' );
    });

    // Also update when country selector changes via keyboard
    $('#country').keyup( function() {
        $('#state').toggle( $(this).val() == 'US' );
        $('#province').toggle( $(this).val() == 'CA' );
    });

});
```

The code is working, but you want to simplify it so there's not so much duplicate code.

Why handle both the `change` and `keyup` events? Many websites handle only the `change` event on a select list. This works fine if you make a selection with the mouse, but if you click the select list and then use the up and down arrow keys to select among the options, nothing happens: keystrokes in a select list do not fire the `change` event. If you also handle the `keyup` event, the select list will respond to the arrow keys, providing a better experience for keyboard users.

Solution 1

Move the duplicate code into a function, and call the function both at load time and in response to the event. Use jQuery's `.bind()` method to wire up both event handlers at the same time. And save data used more than once in variables:

```
$(document).ready( function() {

    var $country = $('#country');

    function setVisibility() {
        var value = $country.val();
        $('#state').toggle( value == 'US' );
```

```
        $('#province').toggle( value == 'CA' );
    }

    setVisibility();
    $country.bind( 'change keyup', setVisibility );
});
```

Solution 2

Use jQuery's event triggering to fire the event immediately after attaching it, along with the `.bind()` trick and local variables from solution 1:

```
$(document).ready( function() {

    $('#country')
        .bind( 'change keyup', function() {
           var value = $(this).val();
           $('#state').toggle( value == 'US' );
           $('#province').toggle( value == 'CA' );
        })
        .trigger('change');

});
```

Discussion

It's standard programming practice in just about any language to take duplicate code and move it into a separate function that can be called from multiple places. Solution 1 follows this approach: instead of repeating the code to set the visibility, it appears once in the `setVisibility()` function. The code then calls that function directly at startup and indirectly when the `change` event is fired.

Solution 2 also uses a common function for both of these cases. But instead of giving the function a name so it can be called directly at startup, the code merely sets the function as the event handler for the `change` event and then uses the `trigger()` method to trigger that same event—thus calling the function indirectly.

These approaches are more or less interchangeable; it's largely a matter of taste which you prefer.

5.4 Formatting Your jQuery Chains

Problem

You have a lengthy jQuery chain that includes methods like `.children()` and `.end()` to operate on several related groups of elements. It's getting hard to tell which operations apply to which elements:

```
$('#box').addClass('contentBox').children(':header')
    .addClass('contentTitle').click(function() {
        $(this).siblings('.contentBody').toggle();
```

```
}).end().children(':not(.contentTitle)')
.addClass('contentBody').end()
.append('<div class="contentFooter"></div>')
.children('.contentFooter').text('generated content');
```

Solution

Put each method call in the chain on its own line, and put the `.` operators at the beginning of each line. Then, indent each part of the chain to indicate where you are switching to different sets of elements.

Increase the indentation when you use methods like `.children()` or `.siblings()` to select different elements, and decrease the indentation when you call `.end()` to return to the previous jQuery selection.

If you're new to jQuery, you'll probably want to read the recipes about basic chaining and `.end()` in Chapter 1:

```
$('#box')
    .addClass('contentBox')
    .children(':header')
        .addClass('contentTitle')
        .click(function() {
            $(this).siblings('.contentBody').toggle();
        })
    .end()
    .children(':not(.contentTitle)')
        .addClass('contentBody')
    .end()
    .append('<div class="contentFooter"></div>')
    .children('.contentFooter')
        .text('generated content');
```

Discussion

By breaking each call out onto its own line, it becomes very easy to scan the code and see what is happening. Using indentation to indicate when you're modifying the set of elements makes it easy to keep track of when destructive operations are occurring and being undone via `.end()`.

This style of indentation results in every call for any given set of elements always being lined up, even if they're not consecutive. For example, it's clear that the wrapper `<div>` has an element prepended and appended to it, even though there are operations on other elements in between.

Putting the `.` operators at the beginning of the lines instead of the end is just a finishing touch: it gives a better visual reminder that these are method calls and not ordinary function calls.

Did jQuery invent chaining? No. jQuery does make very good use of method chaining, but it's something that has been around since the earliest days of JavaScript.

For example, here is a familiar use of chaining with a string object:

```
function htmlEscape( text ) {
    return text
        .replace( '&', '&amp;' )
        .replace( '<', '&lt;' )
        .replace( '>', '&gt;' );
}
```

5.5 Borrowing Code from Other Libraries

Problem

You found a useful function in another JavaScript library and want to use the same technique in your jQuery code. In this case, it's the `.radioClass()` method from the Ext Core library, which adds a class to the matching element(s) and *removes* the same class from all siblings of the matching element(s).

The name `.radioClass()` comes from the behavior of radio buttons in both web applications and desktop apps, where clicking one button selects it and deselects the other buttons in the same radio button group.

The name *radio button* for those input elements comes from the station buttons in old car radios—the mechanical ones where pushing in one button caused all of the other buttons to pop out.

Given this HTML:

```
<div>
    <div id="one" class="hilite">One</div>
    <div id="two">Two</div>
    <div id="three">Three</div>
    <div id="four">Four</div>
</div>
```

you'd like to run code like this:

```
// Add the 'hilite' class to div#three, and
// remove the class from all of its siblings
// (e.g. div#one)

$('#three').radioClass('hilite');
```

You may even want to allow a "multiple-select" radio class:

```
// Add the 'hilite' class to div#two and
// div#four, and remove the class from the
```

```
// other siblings (div#one and div#three)

$('#two,#four').radioClass('hilite');
```

Solution

Write a simple plugin to add the `.radioClass()` method to jQuery:

```
// Remove the specified class from every sibling of the selected
// element(s), then add that class to the selected element(s).
// Doing it in that order allows multiple siblings to be selected.
//
// Thanks to Ext Core for the idea.

jQuery.fn.radioClass = function( cls ) {
    return this.siblings().removeClass(cls).end().addClass(cls);
};
```

This is a short enough function that it's not too hard to follow as a one-liner, but indenting the code as described in Recipe 5.4 makes it completely clear how it works:

```
jQuery.fn.radioClass = function( cls ) {
    return this                 // Start chain, will return its result
        .siblings()             // Select all siblings of selected elements
            .removeClass(cls)   // Remove class from those siblings
        .end()                  // Go back to original selection
        .addClass(cls);         // Add class to selected elements
};
```

Discussion

The composer Igor Stravinsky is reported to have said, "Good composers borrow; great composers steal." He apparently stole the quote from T.S. Eliot, who wrote, "Immature poets imitate; mature poets steal."

Good ideas come from many places, and other JavaScript libraries are chock-full of good code and ideas. If there is code in another open source library that you can use or that you can translate to work with jQuery, you're free to do that—if you respect the other author's copyright and license.

For information on open source and free software, see the following sites:

- *http://www.opensource.org/*
- *http://www.fsf.org/*

You may not even need the actual code in a case like this one, where the implementation is very simple and just the *idea* of having a "radio class" method is the missing link. While not required, it's a good courtesy to give credit to the source of the idea.

Whether the idea comes from elsewhere or is something you thought of yourself, in a surprising number of cases you can write a useful jQuery plugin in one or a few lines of code.

What Is jQuery.fn, and Why Do jQuery Plugins Use It?

`jQuery.fn` is a reference to the same object as `jQuery.prototype`. When you add a function to the `jQuery.fn` object, you're really adding it to `jQuery.prototype`.

When you create a jQuery object with `jQuery()` or `$()`, you're actually calling `new jQuery()`. (The jQuery code automatically does the `new` for you.) As with any other JavaScript constructor, `jQuery.prototype` provides methods and default properties for the objects returned by each `new jQuery()` call. So, what you're really doing when you write a `jQuery.fn` plugin is traditional object-oriented JavaScript programming, adding a method to an object using the constructor's prototype.

Then why does `jQuery.fn` exist at all? Why not just use `jQuery.prototype` like any other object-oriented JavaScript code? It's not just to save a few characters.

The very first version of jQuery (long before 1.0) didn't use JavaScript's `prototype` feature to provide the methods for a jQuery object. It *copied* references to every property and method in `jQuery.fn` (then called `$.fn`) into the jQuery object by looping through the object.

Since this could be hundreds of methods and it happened every time you called `$()`, it could be rather slow. So, the code was changed to use a JavaScript prototype to eliminate all the copying. To avoid breaking plugins that already used `$.fn`, it was made an alias of `$.prototype`:

```
$.fn = $.prototype;
```

So that's why `jQuery.fn` exists today—because plugins used `$.fn` in early 2006!

5.6 Writing a Custom Iterator

Problem

You've selected multiple elements into a jQuery object, and you need to iterate through those elements with a pause between each iteration, for example, to reveal elements one by one:

```
<span class="reveal">Ready? </span>
<span class="reveal">On your mark! </span>
<span class="reveal">Get set! </span>
<span class="reveal">Go!</span>
```

You tried using `each()`, but of course that revealed the elements all at once:

```
$('.reveal').each( function() {
    $(this).show();
});
```

```
// That was no better than this simpler version:
$('.reveal').show();
```

Solution

Write a custom iterator that uses `setTimeout()` to delay the callbacks over time:

```
// Iterate over an array (typically a jQuery object, but can
// be any array) and call a callback function for each
// element, with a time delay between each of the callbacks.
// The callback receives the same arguments as an ordinary
// jQuery.each() callback.
jQuery.slowEach = function( array, interval, callback ) {
    if( ! array.length ) return;
    var i = 0;
    next();

    function next() {
        if( callback.call( array[i], i, array[i] ) !== false )
            if( ++i < array.length )
                setTimeout( next, interval );
    }

    return array;
};
// Iterate over "this" (a jQuery object) and call a callback
// function for each element, with a time delay between each
// of the callbacks.
// The callback receives the same arguments as an ordinary
// jQuery(...).each() callback.
jQuery.fn.slowEach = function( interval, callback ) {
    return jQuery.slowEach( this, interval, callback );
};
```

Then simply change your `.each()` code to use `.slowEach()` and add the timeout value:

```
// Show an element every half second
$('.reveal').slowEach( 500, function() {
    $(this).show();
});
```

Discussion

jQuery's `.each()` method is not rocket science. In fact, if we strip the jQuery 1.3.2 implementation down to the code actually used in the most typical use (iterating over a jQuery object), it's a fairly straightforward loop:

```
jQuery.each = function( object, callback ) {
    var value, i = 0, length = object.length;
    for(
        value = object[0];
        i < length  &&  callback.call( value, i, value ) !== false;
        value = object[++i]
    ) {}
```

```
    return object;
};
```

That could also be coded in a more familiar way:

```
jQuery.each = function( object, callback ) {
    for(
        var i = 0, length = object.length;
        i < length;
        ++i
    ) {
        var value = object[i];
        if( callback.call( value, i, value ) === false )
            break;
    }

    return object;
};
```

We can write similar functions to iterate over arrays or jQuery objects in other useful ways. A simpler example than `.slowEach()` is a method to iterate over a jQuery object in reverse:

```
// Iterate over an array or jQuery object in reverse order
jQuery.reverseEach = function( object, callback ) {
    for( var value, i = object.length;  --i >= 0; ) {
        var value = object[i];
        console.log( i, value );
        if( callback.call( value, i, value ) === false )
            break;
    }
};
// Iterate over "this" (a jQuery object) in reverse order
jQuery.fn.reverseEach = function( callback ) {
    jQuery.reverseEach( this, callback );
    return this;
};
```

This doesn't attempt to handle all of the cases that `.each()` handles, just the ordinary case for typical jQuery code.

Interestingly enough, a custom iterator may not use a loop at all. `.reverseEach()` and the standard `.each()` both use fairly conventional loops, but there's no explicit JavaScript loop in `.slowEach()`. Why is that, and how does it iterate through the elements without a loop?

JavaScript in a web browser does not have a `sleep()` function as found in many languages. There's no way to pause script execution like this:

```
doSomething();
sleep( 1000 );
doSomethingLater();
```

Instead, as with any asynchronous activity in JavaScript, the `setTimeout()` function takes a callback that is called when the time interval elapses. The `.slowEach()` method

increments the "loop" variable `i` in the `setTimeout()` callback, using a closure to preserve the value of that variable between "iterations." (See Recipe 5.2 for a discussion of closures.)

Like `.each()`, `.slowEach()` operates directly on the jQuery object or array you give it, so any changes you make to that array before it finishes iterating will affect the iteration. Unlike `.each()`, `.slowEach()` is asynchronous (the calls to the callback function happen *after* `.slowEach()` returns), so if you change the jQuery object or its elements after `.slowEach()` returns but before all the callbacks are done, that can also affect the iteration.

5.7 Toggling an Attribute

Problem

You need a way to toggle all of the checkmarks in a group of checkboxes. Each checkbox should be toggled independently of the others.

Solution

Write a `.toggleCheck()` plugin that works like the `.toggle()` and `.toggleClass()` methods in the jQuery core to allow you to set, clear, or toggle a checkbox or group of checkboxes:

```
// Check or uncheck every checkbox element selected in this jQuery object
// Toggle the checked state of each one if check is omitted.

jQuery.fn.toggleCheck = function( check ) {
    return this.toggleAttr( 'checked', true, false, check );
};
```

Then you can enable a group of buttons:

```
$('.toggleme').toggleCheck( true );
```

or disable them:

```
$('.toggleme').toggleCheck( false );
```

or toggle them all, each one independent of the rest:

```
$('.toggleme').toggleCheck();
```

This `.toggleCheck()` method is built on top of a more general-purpose `.toggleAttr()` method that works for any attribute:

```
// For each element selected in this jQuery object,
// set the attribute 'name' to either 'onValue' or 'offValue'
// depending on the value of 'on. If 'on' is omitted,
// toggle the attribute of each element independently
// between 'onValue' and 'offValue'.
// If the selected value (either 'onValue' or 'offValue') is
```

```
// null or undefined, remove the attribute.
jQuery.fn.toggleAttr = function( name, onValue, offValue, on ) {

    function set( $element, on ) {
        var value = on ? onValue : offValue;
        return value == null ?
            $element.removeAttr( name ) :
            $element.attr( name, value );
    }
    return on !== undefined ?
        set( this, on ) :
        this.each( function( i, element ) {
            var $element = $(element);
            set( $element, $element.attr(name) !== onValue );
        });
};
```

Why go to the trouble of building something so general-purpose? Now we can write similar togglers for other attributes with almost no effort. Suppose you need to do the same thing as `.toggleCheck()`, but now you're enabling and disabling input controls. You can write a `.toggleEnable()` in one line of code:

```
// Enable or disable every input element selected in this jQuery object.
// Toggle the enable state of each one if enable is omitted.

jQuery.fn.toggleEnable = function( enable ) {
    return this.toggleAttr( 'disabled', false, true, enable );
};
```

Note how the `onValue` and `offValue` parameters let us swap the `true` and `false` attribute values, making it easy to talk about "enabling" the element instead of the less intuitive "disabling" that the `disabled` attribute provides normally.

As another example, suppose we need to toggle a `foo` attribute where its "on" state is the string value `bar`, and its "off" state is to remove the attribute. That's another one-liner:

```
// Add or remove an attribute foo="bar".
// Toggle the presence of the attribute if add is omitted.

jQuery.fn.toggleFoo = function( add ) {
    return this.toggleAttr( 'foo', 'bar', null, add );
};
```

Discussion

It's always good to beware of feeping creaturism (aka creeping featurism). If all we really needed were to toggle checkboxes, we could code the whole thing like this:

```
jQuery.fn.toggleCheck = function( on ) {
    return on !== undefined ?
        this.attr( 'checked', on ) :
        this.each( function( i, element ) {
            var $element = $(element);
            $element.attr( 'checked', ! $element.attr('checked') );
```

```
        });
};
```

That is a bit simpler than our `.toggleAttr()` method, but it's only useful for the `checked` attribute and nothing else. What would we do if we later needed that `.toggleEnable()` method? Duplicate the whole thing and change a few names?

The extra work in `.toggleAttr()` buys us a lot of flexibility: we now can write a whole family of attribute togglers as straightforward one-liners.

Check the documentation for the version of jQuery you're using before writing new utility methods like this. It's always possible that similar methods could be added to future versions of jQuery, saving you the trouble of writing your own.

5.8 Finding the Bottlenecks

Problem

Your site is too slow to load or too slow to respond to clicks and other user interaction, and you don't know why. What part of the code is taking so much time?

Solution

Use a profiler, either one of the many available ones or a simple one you can code yourself.

Discussion

A profiler is a way to find the parts of your code that take the most time. You probably already have at least one good JavaScript profiler at your fingertips. Firebug has one, and others are built into IE 8 and Safari 4. These are all function profilers: you start profiling, interact with your page, and stop profiling, and then you get a report showing how much time was spent in each function. That may be enough right there to tell you which code you need to speed up.

There are also some profilers specific to jQuery that you can find with a web search for *jquery profiler*. These let you profile selector performance and look more deeply at jQuery function performance.

For really detailed profiling, where you need to analyze individual sections of code smaller than the function level, you can write a simple profiler in just a few lines of code. You may have coded this ad hoc classic:

```
var t1 = +new Date;
// ... do stuff ...
var t2 = +new Date;
alert( ( t2 - t1 ) + ' milliseconds' );
```

The `+new Date` in this code is just a simpler way of coding the more familiar `new Date().getTime()`: it returns the current time in milliseconds.

Why does it work? Well, the `new Date` part is the same: it gives you a `Date` object representing the current time. (The `()` are optional, as there are no arguments.) The `+` operator converts that object to a number. The way JavaScript converts an object to a number is by calling the object's `.valueOf()` method. And the `.valueOf()` method for a `Date` object happens to be the same thing as `.getTime()`, giving the time in milliseconds.

We can make something more general-purpose and easier to use with only 15 lines of code:

```
(function() {

    var log = [], first, last;

    time = function( message, since ) {
        var now = +new Date;
        var seconds = ( now - ( since || last ) ) / 1000;
        log.push( seconds.toFixed(3) + ': ' + message + '<br />' );
        return last = +new Date;
    };

    time.done = function( selector ) {
        time( 'total', first );
        $(selector).html( log.join('') );
    };

    first = last = +new Date;
})();
```

Now we have a `time()` function that we can call as often as we want to log the elapsed time since the last `time()` call (or, optionally, since a specific prior time). When we're ready to report the results, we call `time.done()`. Here's an example:

```
// do stuff
time( 'first' );
// do more stuff
time( 'second' );
// and more
time( 'third' );
time.done( '#log' );
```

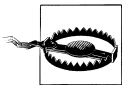
That JavaScript code requires this HTML code to be added to your page:

```
<div id="log">
</div>
```

After the code runs, that `<div>` would get filled with a list like this:

0.102 first
1.044 second
0.089 third
1.235 total

We can see that the largest amount of time is being spent between the `time('first')` and `time('second')` calls.



Beware of Firebug! If you have Firebug enabled on the page you are timing, it can throw off the results considerably. JavaScript's `eval()` function, which jQuery 1.3.2 and earlier use to evaluate downloaded JSON data, is affected to an extreme degree: an array of 10,000 names and addresses in the format from Recipe 5.11 takes 0.2 seconds in Firefox normally, but *55 seconds* with Firebug's Script panel enabled. Later versions of jQuery use `Function()` for this, which isn't affected by Firebug.

If Firebug affects your page as badly as that and if you can't find a workaround, you may want to detect Firebug and display a warning:

```
<div id="firebugWarning" style="display:none;">
    Your warning here
</div>

$(document).ready( function() {
    if( window.console && console.firebug )
        $('#firebugWarning').show();
});
```

For many optimization exercises, this code may be sufficient. But what if the code we need to test is inside a loop?

```
for( var i = 0; i < 10; ++i ) {
    // do stuff
    time( 'first' );
    // do more stuff
    time( 'second' );
    // and more
    time( 'third' );
}
time.done( '#log' );
```

Now our little profiler will list those first, second, and third entries 10 times each! That's not too hard to fix—we just need to accumulate the time spent for each specific message label when it's called multiple times:

```
(function() {

    var log = [], index = {}, first, last;

    // Accumulate seconds for the specified message.
    // Each message string has its own total seconds.
    function add( message, seconds ) {
        var i = index[message];
        if( i == null ) {
            i = log.length;
            index[message] = i;
            log[i] = { message:message, seconds:0 };
        }
        log[i].seconds += seconds;
    }

    time = function( message, since ) {
        var now = +new Date;
        add( message, ( now - ( since || last ) ) / 1000 );
        return last = +new Date;
    }

    time.done = function( sel ) {
        time( 'total', first );
        $(sel).html(
            $.map( log, function( item ) {
                return(
                    item.seconds.toFixed(3) +
                    ': ' +
                    item.message + '<br />'
                );
            }).join('')
        );
    };

    first = last = +new Date;
})();
```

With this change, we'll get useful results from that loop:

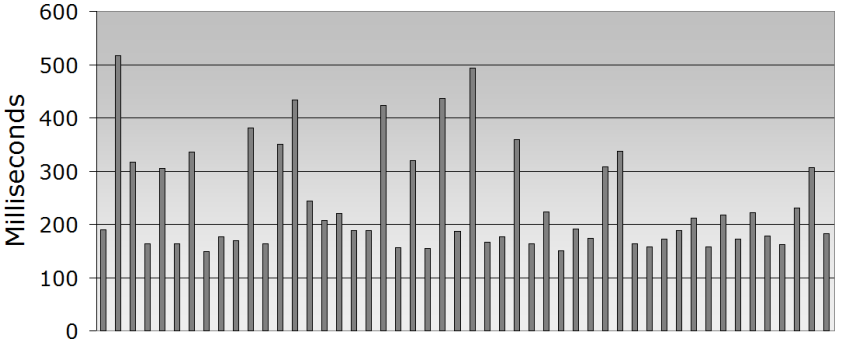
```
0.973 first
9.719 second
0.804 third
11.496 total
```

When Timing Test Results Vary

When you run timing tests on a web page, you won't get the same result every time. In fact, the timing results will probably vary quite a bit if you reload a page or rerun a test multiple times.

What should you do to get the "real" number? Average the results?

Probably not. Here's a chart of the `fillTable()` timing from Recipe 5.11 for 50 consecutive runs, taken about 10 seconds apart:



There's a distinct pattern here: a large majority of runs in the 150–200 millisecond range, with a small number of scattered runs taking longer. It seems likely that something around 175 milliseconds is the real timing, and the runs taking much longer were affected by other processes on the machine.

It's also possible that some of the longer runs are caused by garbage collection in the browser. It would be hard to distinguish that from time taken by other processes, so the most practical thing is probably just to disregard these outliers.

5.9 Caching Your jQuery Objects

Problem

You're logging the various properties of the `event` object for a `mousemove` event, and the code lags behind because it uses `$('.classname')` selectors to find and update table cells with the event data.

Your page contains this HTML code for the log:

```
<table id="log">
    <tr><td>Client X:</td><td class="clientX"></td></tr>
    <tr><td>Client Y:</td><td class="clientY"></td></tr>
    <tr><td>Page X:</td><td class="pageX"></td></tr>
    <tr><td>Page Y:</td><td class="pageY"></td></tr>
    <tr><td>Screen X:</td><td class="screenX"></td></tr>
    <tr><td>Screen Y:</td><td class="screenY"></td></tr>
</table>
```

and this JavaScript code:

```
$('html').mousemove( function( event ) {
    $('.clientX').html( event.clientX );
    $('.clientY').html( event.clientY );
    $('.pageX').html( event.pageX );
    $('.pageY').html( event.pageY );
    $('.screenX').html( event.screenX );
    $('.screenY').html( event.screenY );
});
```

The page also contains a large number (thousands!) of other DOM elements. In a simpler test page, the code performs fine, but in this complex page it is too slow.

Solution

Cache the jQuery objects returned by the `$(...)` calls, so the DOM queries only have to be run once:

```
var
    $clientX = $('.clientX'),
    $clientY = $('.clientY'),
    $pageX = $('.pageX'),
    $pageY = $('.pageY'),
    $screenX = $('.screenX'),
    $screenY = $('.screenY');
$('html').mousemove( function( event ) {
    $clientX.html( event.clientX );
    $clientY.html( event.clientY );
    $pageX.html( event.pageX );
    $pageY.html( event.pageY );
    $screenX.html( event.screenX );
    $screenY.html( event.screenY );
});
```

You may also be able to speed up those selectors considerably; see the next recipe for ways to do that. But simply calling them once each instead of over and over again may be enough of an improvement right there.

Discussion

One of the classic ways to optimize code is to "hoist" repeated calculations out of a loop so you have to do them only once. Any values that don't change inside the loop should be calculated one time, before the loop starts. If those are expensive calculations, the loop will then be much faster.

This works just as well when the "loop" is a series of frequently fired events such as `mousemove` and the "calculation" is a jQuery selector. Hoisting the selector out of the event handler makes the event handler respond faster.

Of course, if you're calling multiple selectors inside a loop, that will also benefit from moving them outside the loop in the same manner.

Why do `$clientX` and the other variable names begin with the `$` character?

`$` doesn't have any special meaning in JavaScript—it's treated just like a letter of the alphabet. It's simply a popular convention in jQuery code to use the `$` prefix as a reminder that the variable contains a reference to a jQuery object and not, say, a DOM element, because a variable name of `$foobar` has a visual resemblance to the jQuery operation `$('#foobar')`.

This is especially helpful when you need to use both a jQuery object and its underlying DOM element, e.g.:

```
var $foo = $('#foo'), foo = $foo[0];
// Now you can use the jQuery object:
$foo.show();
// or the DOM element:
var id = foo.id;
```

5.10 Writing Faster Selectors

Problem

Your code contains a large number of `$('.classname')` selectors. You're caching them as described in the previous recipe, but the selectors are still affecting your page load time. You need to make them faster.

Solution

First, make sure you are using a recent version of jQuery (1.3.2 or later) for faster selector performance in most browsers, especially with class selectors.

If you have control over the HTML page content, change the page to use `id` attributes and `'#xyz'` selectors instead of `class` attributes and `'.xyz'` selectors:

```
<div class="foo"></div>
<div id="bar"></div>

$('.foo')  // Slower
$('#bar')  // Faster
```

If you must use class name selectors, see whether there is a parent element that you can find with a faster ID selector, and then drill down from there to the child elements. For example, using the HTML from the previous recipe:

```
<table id="log">
    <tr><td>Client X:</td><td id="clientX"></td></tr>
    ...
</table>
```

you could use this:

```
$('.clientX')         // Slower
$('td.clientX')       // May be faster
$('#log .clientX')    // May be much faster
$('#log td.clientX')  // Possibly faster in some browsers
```

Beware of selector speed test pages that don't reflect the actual page content you are using. In a very simple page, a simple `$('.clientX')` selector may test out faster than a fancier selector like `$('#log td.clientX')`—even in browsers and jQuery versions where you might expect the class selector to be slow.

That's just because the more complicated selector takes more time to set up, and in a simple page that setup time may dominate performance.

The test page for this recipe deliberately contains a very large number of elements to provoke selector performance problems that only show up in large pages.

Neither one, of course, shows exactly what any selector's performance will be in *your* page. The only way to be sure which selector is fastest in a particular page is to test each in that page.

Discussion

It's easy to forget that an innocent-looking call like `$('.clientX')` may take considerable time. Depending on the browser and version of jQuery, that selector may have to make a list of every DOM element in your page and loop through it looking for the specified class.

jQuery versions prior to 1.3 use this slow method in *every* browser. jQuery 1.3 introduced the Sizzle selector engine, which takes advantage of faster DOM APIs in newer browsers such as `getElementsByClassName()` and `querySelectorAll()`.

However, for most websites you'll probably need to support IE 7 for some time to come, and class selectors are slow in IE 7 when you have a complex page.

If you can use it, selecting by ID as in `$('#myid')` is generally very fast in all browsers, because it simply uses a single call to the `getElementById()` API.

It also helps to narrow down the number of elements that need to be searched, either by specifying a parent element, by making the class selector more specific with a tag name, or by combining those and other tricks.

5.11 Loading Tables Faster

Problem

You're loading a JSON data object with 1,000 names and addresses and using jQuery to create a table with this data. It takes 5–10 seconds to create the table in IE 7—and that's not even counting the download time.

Your JSON data is in this format:

```
{
    "names": [
        {
            "first": "Azzie",
            "last": "Zalenski",
            "street": "9554 Niemann Crest",
            "city": "Quinteros Divide",
            "state": "VA",
            "zip": "48786"
        },
        // and repeat for 1000 names
    ]
}
```

Your JavaScript code is as follows:

```
// Return a sanitized version of text with & < > escaped for HTML
function esc( text ) {
    return text
        .replace( '&', '&amp;' )
        .replace( '<', '&lt;' )
        .replace( '>', '&gt;' );
}
$(document).ready( function() {

    function fillTable( names ) {
        $.each( names, function() {
            $('<tr>')
                .append( $('<td>').addClass('name').html(
                    esc(this.first) + ' ' + esc(this.last)
                ) )
                .append( $('<td>').addClass('address').html(
                    esc(this.street) + '<br />' +
                    esc(this.city) + ', ' +
                    esc(this.state) + ' ' + esc(this.zip)
                ) )
                .appendTo('#nameTable');
        });
    }

    $.getJSON( 'names/names-1000.json', function( json ) {
        fillTable( json.names );
    });
});
```

And you have this HTML code in your document:

```
<table id="nameTable">
</table>
```

It works fine, resulting in the browser display shown in Figure 5-1.

Arica Hence	5473 Brallier Crossing Bejar Centers, MA 37967
Vicente Hofmann	4070 Cree Pike Gosier Cove, LA 67602
Renna Pastorius	4666 Moorer Tunnel Marmo Centers, PO 53702
Exie Duca	7139 Langenfeld Court Albrecht, RE 24425
Sean Pickerel	9956 Urquidez Rest Iveson Islands, EN 05425
Justa Ocasio	3463 Lair Terrace Stanislawski, SE 80277
Sommer Lafortune	9101 Palmateer Road

Figure 5-1. Browser output for name table

It's just much too slow.

Solution

Combine several optimizations:

- Insert a single `<table>` or `<tbody>` instead of multiple `<tr>` elements
- Use `.innerHTML` or `.html()` instead of DOM manipulation
- Build an array with `a[++i]` and `.join()` it instead of string concatenation
- Use a bare-metal `for` loop instead of `$.each`
- Reduce name lookups

The result is this new version of the code (using the same `esc()` function as before):

```
$(document).ready( function() {

    function fillTable( names ) {
        // Reduce name lookups with local function name
        var e = esc;
        //
        var html = [], h = -1;
        html[++h] = '<table id="nameTable">';
        html[++h] = '<tbody>';
        for( var name, i = -1;  name = names[++i]; ) {
            html[++h] = '<tr><td class="name">';
            html[++h] =     e(name.first);
```

```
            html[++h] =     ' ';
            html[++h] =     e(name.last);
            html[++h] = '</td><td class="address">';
            html[++h] =     e(name.street);
            html[++h] =     '<br />';
            html[++h] =     e(name.city);
            html[++h] =     ', ';
            html[++h] =     e(name.state);
            html[++h] =     ' ';
            html[++h] =     e(name.zip);
            html[++h] = '</td></tr>';
        }
        html[++h] = '</tbody>';
        html[++h] = '</table>';

        $('#container')[0].innerHTML = html.join('');
    }

    $.getJSON( 'names/names-1000.json', function( json ) {
        fillTable( json.names );
    });
});
```

The new code requires the HTML code in your document to be changed to the following:

```
<div id="container">
</div>
```

On one test system in IE 7, the new code runs in 0.2 seconds compared with 7 seconds for the original code. That's 35 times faster!

Granted, the code is not as clean and elegant as the original, but your site's visitors will never know or care about that. What they will notice is how much faster your page loads.

Discussion

Sometimes you'll get lucky and find that one specific optimization is all it takes to fix a performance problem. Sometimes, as in this recipe, you'll need several tricks to get the speed you want.

The biggest speed boost in this code comes from inserting a single `<table>` element with all its children in a single DOM operation, instead of inserting a lengthy series of `<tr>` elements one by one. In order to do this, you need to generate the entire table as HTML. That means you need to paste together a large number of strings to build the HTML, which can be very fast or very slow depending on how you do it. And with 1,000 items to loop though, it's worth finding the fastest way to write the loop itself.

You may wonder, "Is this still jQuery code? It looks like plain old JavaScript!" The answer is yes, and yes. It's quite all right to mix and match jQuery code with other JavaScript code. You can use simpler jQuery ways of coding in most of your site, and

when you discover the slow parts, you can either find faster jQuery techniques or use plain old JavaScript as needed for performance.

5.12 Coding Bare-Metal Loops

Problem

You're calling `$.each(array,fn)` or `$(selector).each(fn)` to iterate over thousands of items in your code, and you suspect that all those function calls may be adding to your load time:

```
$.each( array, function() {
    // do stuff with this
});
```

or:

```
$('.lotsOfElements').each( function() {
    // do stuff with this or $(this)
});
```

Solution

Use a `for` loop instead of `.each()`. To iterate over an array, it's hard to beat this loop:

```
for( var item, i = -1; item = array[++i] ) {
    // do stuff with item
}
```

But there is a catch: this loop works only if your array has no "false" elements, that is, elements whose value is `undefined`, `null`, `false`, `0`, or `""`. Even with that restriction, this loop is useful in many common cases, such as iterating over a jQuery object. Just be sure to cache the object in a variable:

```
var $items = $('.lotsOfElements');
for( var item, i = -1; item = $item[++i] ) {
    // do stuff with item (a DOM node)
}
```

It's also common to have JSON data that contains an array of objects as in our example from Recipe 5.11:

```
{
    "names": [
        {
            // ...
            "zip": "48786"
        },
        // and repeat for 1000 names
    ]
}
```

If you know that none of the objects making up the elements of the `names` array will ever be null, it's safe to use the fast loop.

For a more general-purpose loop that works with any array, there is always the classic loop that you'll see in many places:

```
for( var i = 0; i < array.length; i++ ) {
     var item = array[i];
    // do stuff with item
}
```

But you can improve that loop in three ways:

- Cache the array length.
- Use `++i`, which is faster than `i++` in some browsers.
- Combine the test and increment of the loop variable to remove one name lookup.

The result is as follows:

```
for( var i = -1, n = array.length; ++i < n; ) {
    var item = array[i];
    // do stuff with item
}
```

Would it be even faster to use a `while` loop or a `do...while` loop? Probably not. You could rewrite the previous loop as follows:

```
var i = -1, n = array.length;
while( ++i < n ) {
    var item = array[i];
    // do stuff with item
}
```

or:

```
var i = 0, n = array.length;
if( i < n ) do {
    var item = array[i];
    // do stuff with item
}
while( ++i < n );
```

But neither one is any faster than the more readable `for` loop.

To iterate over an object (not an array), you can use a `for..in` loop:

```
for( var key in object ) {
    var item = object[key];
    // do stuff with item
}
```

A Warning About for..in Loops

Never use a `for..in` loop to iterate over a jQuery object or an array of any type. If the array has any custom properties or methods, those will be iterated along with the numeric array elements. For example, this code enumerates a single DOM element, the document body (with `i = 0`):

```
$('body').each( function( i ) { console.log( i ); });
```

This code may look like it would do the same thing, but it enumerates all of the jQuery methods such as `show` and `css` along with the `[0]` element:

```
for( var i in $('body') ) console.log( i ); // BAD
```

Instead, use one of the array loops listed previously.

Even the "safe" use of a `for..in` loop to iterate over an object can get in trouble if any code on your page has modified `Object.prototype` to extend all objects with additional methods or properties. The loop will enumerate those methods or properties along with the ones you want.

Extending `Object.prototype` is strongly discouraged because it breaks so much code. In fact, at least through jQuery 1.3.2, it breaks jQuery itself by causing `each()` to enumerate those added methods or properties. If your code has to work in such an environment, you need to take extra precautions on all your loops, such as testing the `hasOwnProperty()` method of each object property. Unfortunately, these extra tests slow the code down, so you have to choose between speed and robustness.

Discussion

`$(selector).each(fn)` is the customary way to create a jQuery object and iterate over it, but it's not the only way. The jQuery object is an "array-like" object with `.length` and `[0]`, `[1]`, `...`, `[length-1]` properties. Therefore, you can use any of the looping techniques you would use with any other array. And because the jQuery object never contains "false" elements, you can use the fastest `for` loop listed at the beginning of the solution.

If you use the `time()` function from Recipe 5.2 or another profiler to measure loop performance, be sure to test your actual code, not a simplified test case that just runs the loop without the full loop body. The simplified test would miss one potential benefit of the `for` loop: fewer name lookups resulting from less function nesting. See Recipe 5.13 for the details.

5.13 Reducing Name Lookups

Problem

Your code has an inner loop, down inside several levels of nested functions, that runs hundreds or thousands of times. The inner loop calls several global functions, and it references some variables defined in the outer functions or globally.

Each of these references is triggering several name lookups because of the nested functions. It's slowing down your code, but profilers don't show what the problem is, and it isn't obvious from looking at the code that there's a problem!

Solution

Investigate every name that appears in your innermost loop, and figure out how many name lookups it requires. Reduce the number of name lookups by caching object references locally or using fewer nested functions.

Discussion

Closures are a wonderful thing. They make it trivial to capture state information and pass it along to asynchronous functions such as event handlers or timer callbacks. If JavaScript didn't have closures, then every asynchronous callback would need to have a way to pass that state around. Instead, you can simply use a nested function.

The dynamic nature of JavaScript is also a wonderful thing. You can add properties and methods to any object, any time, and the JavaScript runtime will happily chase down those references when you need them.

Put these together, and you can get a lot of name lookups.

The most modern JavaScript interpreters have improved greatly in this area. But if you want your code to run fast in the most popular browsers—such as any version of IE—you still need to worry about the number of name lookups.

Consider this code:

```
// A typical function wrapper to get a local scope
(function() {
    // Find the largest absolute value in an array of numbers
    function maxMagnitude( array ) {
        var largest = -Infinity;
        $.each( array, function() {
            largest = Math.max( largest, Math.abs(this) );
        });
     return largest;
    }
```

```
    // Other code here calls maxMagnitude on a large array
})();
```

Remember that JavaScript looks up a name first in the local scope (function), and if the name isn't found there, it works its way up through the parent nested functions and finally the global scope. Not only does the JavaScript runtime have to look up each name every time you use it, it also has to repeat those lookups when the names are actually defined in parent functions or in the global scope.

So, if this block of code is in the global scope, the `each()` callback does the following name lookups in every iteration:

1. `largest` in local scope [fail]
2. `largest` in `MaxMagnitude()` [success]
3. `Math` in local scope [fail]
4. `Math` in `MaxMagnitude()` [fail]
5. `Math` in anonymous wrapper function [fail]
6. `Math` in global scope [success]
7. `abs` in `Math` object [success]
8. `Math` in local scope [fail]
9. `Math` in `MaxMagnitude()` [fail]
10. `Math` in anonymous wrapper function [fail]
11. `Math` in global scope [success]
12. `max` in `Math` object [success]
13. `largest` in local scope [fail]
14. `largest` in `MaxMagnitude()` [success]

Now rewrite the code as follows:

```
// A typical wrapper to get a local scope
(function() {
    // Find the largest absolute value in an array of numbers
    function maxMagnitude( array ) {
        var abs = Math.abs, max = Math.max;
        var largest = -Infinity;
        for( var i = -1, n = array.length; ++i < n; ) {
            largest = max( largest, abs(array[i]) );
        }
        return largest;
    }
    // Other code here calls maxMagnitude on a large array
})();
```

This not only eliminates the callback function call in every iteration, it also reduces the number of name lookups per iteration by 10 or more. The loop body in this version does these name lookups:

1. largest in local scope [success]
2. abs in local scope [success]
3. max in local scope [success]
4. largest in local scope [success]

That's more than a 70 percent improvement over the first version.

If this code is nested even deeper inside another function, the difference is even greater, since each nested function adds one more lookup for each of the `Math` object lookups.

In this discussion we're omitting the `this` and `array[i]` lookups, as well as the lookups in the `for` loop itself. Those are roughly comparable between the two versions.

In Recipe 5.11, a single name lookup optimization accounts for a 100 ms improvement. That's not a huge difference, but a tenth of a second off your page load time for a one-line code change is good value.

The original code calls `esc()` six times in each loop iteration, for a total of 6,000 calls in the thousand-name test case. These calls are inside three nested functions, and `esc()` is a global function, so it takes four name lookups simply to resolve the function name for each call. That's 24,000 name lookups!

The improved code reduces the function nesting by one, so that cuts it down to 18,000 name lookups (two nested functions and the global scope at 6,000 each), but then it uses one last trick in the innermost function:

```
function fillTable( names ) {
    var e = esc;
    // and now call e() in the inner loop instead of esc()
}
```

Now, the 6,000 calls to `e()` are each resolved in a single name lookup. That's a reduction of 12,000 name lookups. No wonder it knocks a tenth of a second off the load time.

5.14 Updating the DOM Faster with .innerHTML

Problem

You're creating a large block of HTML code and using `$('#mydiv').html(myhtml);` to insert it into the DOM. You've profiled the code and found that the `.html()` method is taking longer than expected.

Solution

Use `$('#mydiv')[0].innerHTML = myhtml;` for faster DOM updates—if you don't require any of the special processing that `.html()` provides.

Discussion

The `.html()` method uses the `.innerHTML` property to actually insert the HTML content into the DOM, but it does quite a bit of preprocessing first. In most cases this won't

matter, but in performance-critical code you can save some time by setting the `.innerHTML` property directly.

It's actually jQuery's internal `.clean()` method that does this processing. If you read the source code for `.clean()`, you'll see that it goes to quite a bit of work to clean up the HTML input.

The easiest way to find most methods in the jQuery source code is to search for the method name with a `:` after it; e.g., to find the `.clean()` method, search for `clean:` in the *uncompressed* jQuery source.

The code in Recipe 5.11 runs afoul of some of this cleanup. That recipe's HTML code contains a large number of `<br />` tags. There's a regular expression in `.clean()` that finds all self-closing tags (tags that end with `/>` and therefore do not require a closing tag) and checks that these tags are indeed in the limited set of HTML tags that can be self-closed. If not, then it converts the HTML to an open–close tag.

For example, if you code `$('#test').html('<div />');`, then this invalid HTML is automatically converted to `$('#test').html('<div></div>');`. This makes coding easier, but if you have a very long HTML string that contains many self-closing tags, `.clean()` has to check them all—even if all those tags are valid like the `<br />` tags in the other recipe.

The `.html()` method replaces any existing content, and it takes care to avoid memory leaks by removing all event handlers that you've attached through jQuery to any of the elements being replaced. If there are any event handlers in the content being replaced, you should stick with `.html()`, or if you just need this event handler cleanup but don't need the other HTML cleanup, you could possibly use `$('#test').empty()[0].innerHTML = myhtml;` to get the event cleanup only.

The bottom line: if you know for sure that your code doesn't require the event cleanup or HTML cleanup that jQuery normally provides, then with caution you can use `.innerHTML` directly. Otherwise, stick with `.html()` for safety.

5.15 Debugging? Break Those Chains

Problem

A chain of jQuery methods is failing somewhere along the way. The HTML code is as follows:

```
<div class="foo">
    before
        <span class="baz" style="display:none;">
            test
        </span>
```

```
    after
</div>
```

and the JavaScript code (part of a button click event handler) is as follows:

```
$('.foo').css({ fontsize: '18px' }).find('.bar').show();
```

But when you run the code, the font size isn't set, and the hidden element isn't shown.

You have Firebug or another JavaScript debugger, but it's hard to trace through the code. How can you tell where in the chain it is failing?

Solution

Break up the chain into individual statements, and store each jQuery object in a variable:

```
// $('.foo').css({ fontsize: '18px' }).find('.bar').show();
var $foo = $('.foo');
$foo.css({ fontsize: '18px' });
var $bar = $foo.find('.bar');
$bar.show();
```

Now you have several debugging options. One is to use the Step Over command in the debugger to single step over each statement and observe your variables and the state of the page after each step.

In this code, you'd want to check `$foo` and `$bar` after their values are assigned. What is the value of the `.length` property of each? That tells you how many DOM elements were selected. Does each object contain the DOM elements you expect? Check the `[0]`, `[1]`, `[2]`, etc., properties to see the DOM elements.

Assuming that `$foo` contains the correct DOM elements, what happens after the `.css()` method is called? With Firebug's CSS Inspector, you'll find that the CSS `font-size` property is unchanged after the method call. Wait a minute! It's `font-size`, not `fontsize`? There's the problem. Checking the docs, you find that the correct way to write this is either of these:

```
$foo.css({ fontSize: '18px' });

$foo.css({ 'font-size': '18px' });
```

That's one problem down, but what about the other one? After `$bar` is assigned, if we look at its `.length` property, we'll see that it is zero. This tells us that we didn't succeed in selecting any elements. A look at the HTML and JavaScript code will then reveal that we simply misspelled the class name.

Now we can incorporate these two fixes back in the original chain:

```
$('.foo').css({ fontSize: '18px' }).find('.baz').show();
```

Another alternative is to use Firebug's logging statements:

```
// $('.foo').css({ fontsize: '18px' }).find('.bar').show();
var $foo = $('.foo');
```

```
console.log( $foo );
$foo.css({ fontsize: '18px' });
console.log( $foo.css('fontsize') );
var $bar = $foo.find('.bar');
console.log( $bar );
$bar.show();
```

These `console.log()` calls will reveal that `$bar` doesn't have any elements selected, although we've fallen into a trap on the call that attempts to log the font size: we misspelled `fontSize` in the `console.log()` call as well!

This is where combining multiple debugging techniques helps: log those variables, use Firebug's inspectors, read and reread your source code, and have someone else look at the problem too.

Discussion

jQuery's chaining helps make it easy to write concise code, but it can get in the way when debugging, because it is hard to step through the individual steps of the chain and see their results. Breaking up the chain into individual statements, even on a temporary basis while debugging, makes this task easier.

5.16 Is It a jQuery Bug?

Problem

You're calling some jQuery code to show a hidden element and set its HTML content after a time delay using `setTimeout()`:

```
function delayLog( text ) {
    setTimeout( "$('#log').show().html(text)", 1000 );
}
// ... and somewhere else in the code ...
delayLog( 'Hello' );
```

The `.show()` call works, but the `.html(text)` call fails. The Firebug console reports that the `text` variable is undefined. The same jQuery code works when you don't call it from `setTimeout()`. Is there a problem using jQuery with `setTimeout()`?

Solution

One way to find out whether jQuery is the source of a problem is to replace your jQuery code with other JavaScript code that doesn't use jQuery. In this example, we can replace the jQuery code with a simple `alert()`:

```
function delayLog( text ) {
    setTimeout( "alert(text)", 1000 );
}
```

When we try this version of the code, the same problem occurs: there is no alert, and Firebug again reports that `text` is undefined.

This doesn't identify the problem, but it narrows it down a lot. It clearly isn't jQuery (unless the mere presence of the jQuery library is interfering with your page, but you can rule that out by running the code in a simple test page that doesn't include jQuery). So, it must be something wrong with this code itself, most likely to do with the way we're using `setTimeout()`.

Indeed, the problem here is that when a string argument is passed to `setTimeout()`, it is executed in the *global scope*, i.e., as if the code were located outside of any function. The easiest way to fix it is to use a local function for the callback instead of a text string:

```
function delayLog( text ) {
    setTimeout( function() {
        alert(text);
    }, 1000 );
}
```

Unlike code in a string, a nested function has full access to the outer function's variables and parameters. So, this code will alert the text as expected.

And finally, here is a corrected version of the original jQuery code:

```
function delayLog( text ) {
    setTimeout( function() {
        $('#log').show().html(text);
    }, 1000 );
}
```

Discussion

When debugging, if you aren't sure what is causing a problem, finding out where the problem *isn't* can help you track it down. The purpose of this recipe isn't to help you troubleshoot `setTimeout()` problems—after all, this is a jQuery book, not a general JavaScript book—but to help you focus your debugging efforts by quickly ruling out (or confirming!) jQuery as the source of the problem.

5.17 Tracing into jQuery

Problem 1

You're using the Step Into feature in Firebug or another JavaScript debugger to try to step through the jQuery code to see what it actually does when you make a jQuery call. But when you step into the jQuery code, it's all mashed into one long, unreadable line of code and you can't step through it:

```
(function(){var l=this,g,y=l.jQuery,p=l.$,o=l.jQuery=l.$=function(E,F)...
```

Solution 1

You're using the *minified* version of jQuery. Instead, load the *uncompressed* version of jQuery into your page for testing.

If you are loading the code from the Google Ajax Libraries API with a `<script>` tag, change it like this:

```
<!-- Comment out the minified jQuery -->
<!--
<script type="text/javascript"
  src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js
"></script>
-->
<!-- Use the uncompressed version for testing -->
<script type="text/javascript"
  src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.js"></script>
```

If you're using Google's JavaScript loader, change it like this:

```
// Comment out the minified jQuery
// google.load( 'jquery', '1.3.2' );
// Use the uncompressed version for testing
google.load( 'jquery', '1.3.2', { uncompressed:true });
```

Now you will be able to step into and through the jQuery code.

Problem 2

Having fixed that problem, you want to learn how jQuery's `.html()` and `.show()` methods work. So, you are trying to trace into this code in the debugger:

```
$('#test').html( 'test' ).show();
```

But when you use the Step Into command, the debugger goes into the jQuery constructor instead of either of the methods that you're interested in.

Solution 2

The previous line of code contains *three* function calls: a call to the jQuery (`$`) constructor followed by calls to the `.html()` and `.show()` methods. The Step Into command steps into the first of those calls, the constructor.

At that point you can immediately do a Step Out followed by another Step In. This steps you out of the jQuery constructor (thus putting you back in the *middle* of the original line of code) and then into the `.html()` method.

To get to the `.show()` method, use another pair of Step Out and Step In commands. Each time you do this, you'll work your way one step further through the jQuery chain.

If this gets tedious, break the chain as described in Recipe 5.15, and add `debugger;` statements wherever you want to stop. If you want to trace into the `.show()` method, you can change the code to the following:

```
var $test = $('#test');
$test.html( 'test' );
debugger;
$test.show();
```

Now when the code stops on the `debugger;` statement, you can just use Step In (twice, first to step to the `$test.show();` statement and then to step *into* that function call).

You could use Step Over to step from the `debugger;` statement to the next line, since after all you're not yet stepping "into" anything, but it's easier to click Step In (or hit the F11 key in Windows) twice, and it works just as well. Or, instead of the `debugger;` statement, you can set a breakpoint on the `$test.show()` line itself, and then a single Step In will go into the code for the `.show()` method.

Discussion

The minified version of jQuery is great for production use but not so good for development. It collapses all of the code into one or two lines, making it nearly impossible to step through the code in a debugger. Also, the common use of chained methods makes it more difficult to step into jQuery methods. Using the tips in this recipe, you can easily trace through the jQuery code in the debugger, whether to chase down a bug or to learn how the code works.

Do not let your test-driven friends talk you out of using a debugger! Even if you find most of your bugs through unit testing and other means, one of the best ways to learn about a piece of code is to step through it in the debugger and study its variables and properties as you go.

After all, as you read code, you have to step through it in your head and form a mental model of what its variables contain. Why not let the computer step through the code and *show* you what's in those variables?

5.18 Making Fewer Server Requests

Problem

You're including jQuery and a number of plugins in your page. The sheer number of server requests is slowing down your page load time:

```
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript" src="superfish.js"></script>
<script type="text/javascript" src="cycle.js"></script>
<script type="text/javascript" src="history.js"></script>
<script type="text/javascript" src="hoverintent.js"></script>
<script type="text/javascript" src="jcarousel.js"></script>
```

```
<script type="text/javascript" src="thickbox.js"></script>
<script type="text/javascript" src="validate.js"></script>
```

After the page loads, you are downloading some JSON data using `$.getJSON()`, thus adding yet another server request:

```
$(document).ready( function() {
    $.getJSON( 'myjson.php?q=test', function( json ) {
        $('#demo').html( json.foo );
    });
});
```

`myjson.php` is a script on your server that returns JSON data like this:

```
{
    "foo": "bar"
}
```

Solution

Load jQuery from Google's Ajax library, and combine all your plugins into a single file:

```
<script type="text/javascript"

  src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>

<script type="text/javascript" src="plugins.js">
</script>
```

Or, combine *all* of the JavaScript code you use most frequently (jQuery, plugins, and your own code) into a single file:

```
<script type="text/javascript" src="allmyscripts.js"></script>
```

Either way, it also helps to *minify* the `.js` files (remove comments and extra whitespace) to reduce their size. And make sure your server is using gzip compression on the files it downloads.

For the JSON data, since this page is generated by your own server application, you can "burn" the JSON data directly into the HTML page as it's generated, using a `<script>` tag:

```
<script type="text/javascript">
    var myJson = {
        "foo": "bar"
    };
</script>
```

The highlighted portion of that script tag is identical to the JSON data downloaded by `myjson.php` in the original code. In most server languages it should be easy to include the content in this way.

Now the jQuery code to use the JSON data is simply:

```
$(document).ready( function() {
    $('#demo').html( myJson.foo );
});
```

This eliminates one more server request.

Discussion

One of the keys to fast page loading is to simply minimize the number of HTTP requests. Making requests to different servers can also help. Browsers will make only a small number of simultaneous downloads from any single domain (or subdomain), but if you download some of your files from a different domain, the browser may download them in parallel as well.

Pointing different `<script>` tags to different domains may allow them to be downloaded in parallel, but it doesn't affect the order of *execution*. `<script>` tags are executed in the order they appear in the HTML source.

You can combine JavaScript files by hand by simply copying and pasting them into one big file. This is inconvenient for development but does speed up downloading.

There are a number of file combiner/minifiers available for various server languages.

Ruby on Rails:

- Bundle-fu
- AssetPackager
- The packager built into Rails 2.0

PHP:

- Minify

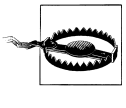
Python:

- JSCompile

Java:

- YUI Compressor

In addition to JavaScript code, check your CSS for multiple `.css` files. Some of the tools listed can merge your `.css` files into a single download, just as they do for `.js` files.



At one time, "packing" JavaScript was all the rage. This not only removes comments and whitespace but also rewrites all of the JavaScript code so that it's not even JavaScript code anymore. Packing requires an unpacking step at runtime—every time the page loads, even if the JavaScript code is already cached. Because of this, packing has fallen out of favor, and "minifying" the code (removing comments and whitespace) is recommended instead, combined with gzip compression. Much of the benefit of packing comes from removing duplicate strings, and gzip does that for you anyway.

5.19 Writing Unobtrusive JavaScript

Problem

You have a page with inline event handler attributes creating a hover effect for a menu.

Your content (HTML), presentation (CSS), and behavior (JavaScript) are all mixed up, making it hard to maintain each on their own and resulting in duplicate JavaScript and style settings:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <meta http-equiv="Content-Language" content="en-us" />
    <title>Menu Demo</title>

    <style type="text/css">
        .menu {
            background-color: #ccc;
            list-style: none;
            margin: 0;
            padding: 0;
            width: 10em;
        }
        .menu li {
            margin: 0;
            padding: 5px;
        }
        .menu a {
            color: #333;
        }
    </style>
</head>
<body>
<ul class="menu">
    <li onmouseover="this.style.backgroundColor='#999';"
        onmouseout="this.style.backgroundColor='transparent';">
        <a href="download.html">Download</a>
    </li>
    <li onmouseover="this.style.backgroundColor='#999';"
        onmouseout="this.style.backgroundColor='transparent';">
        <a href="documentation.html">Documentation</a>
    </li>
    <li onmouseover="this.style.backgroundColor='#999';"
        onmouseout="this.style.backgroundColor='transparent';">
        <a href="tutorials.html">Tutorials</a>
    </li>
</ul>
</body>
</html>
```

Solution

Replace inline JavaScript with jQuery event handlers, and add/remove classes instead of manipulating the `backgroundColor` style directly:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <meta http-equiv="Content-Language" content="en-us" />
    <title>Menu Demo</title>

    <style type="text/css">
        .menu {
            background-color: #ccc;
            list-style: none;
            margin: 0;
            padding: 0;
            width: 10em;
        }
        .menu li {
            margin: 0;
            padding: 5px;
        }
        .menu a {
            color: #333;
        }
        .menuHover {
            background-color: #999;
        }
    </style>

    <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.js">
    </script>

    <script type="text/javascript">

        $(document).ready( function() {
            $('li').hover(
                function() {
                    $(this).addClass('menuHover');
                },
                function() {
                    $(this).removeClass('menuHover');
                });
        });

    </script>
</head>
<body>

<ul class="menu">
    <li><a href="download.html">Download</a></li>
```

```
    <li><a href="documentation.html">Documentation</a></li>
    <li><a href="tutorials.html">Tutorials</a></li>
</ul>
</body>
</html>
```

We've removed the inline event handlers and replaced them with jQuery event handlers, separating the content and behavior. Now if we want to add more menu items, we don't have to copy and paste the same batch of event handlers; instead, the event handler will automatically be added.

We have also moved the style rules for the hover effect into a CSS class, separating the behavior and presentation. If we want to change the styling for the hover effect later, we can just update the stylesheet instead of having to modify the markup.

Discussion

While an "all in one" HTML file with `on*event*` attributes works fine in a small, simple page, it doesn't scale up very well. As your pages get more complex, separating presentation and behavior makes the code easier to maintain.

We didn't do it in this simple example, but if you have multiple pages using the same JavaScript or CSS code, move that code to a common `.js` or `.css` file. That way it will be downloaded into the browser cache once, instead of being re-sent on every page load. As a result, once one of your pages has been visited, the rest will load faster.

5.20 Using jQuery for Progressive Enhancement

Problem

You want to build a site that allows simple task management with a great user experience using animations and Ajax, but you also want to support users who have JavaScript disabled.

Solution

You can build the site to work without all the flashiness and then unobtrusively add the JavaScript functionality:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <meta http-equiv="Content-Language" content="en-us" />
    <title>Task List</title>

    <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.js">
    </script>
```

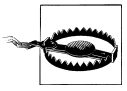
```
    <script type="text/javascript">

        $(document).ready( function() {
            var url = $('form').attr('action');
            $(':checkbox').click(function() {
                $.post(url, this.name + '=1');
                $(this).parent().slideUp(function() {
                    $(this).remove();
                });
            });
            $(':submit').hide();
        });

    </script>
</head>
<body>
<form method="post" action="tasklist.html">
    <ul>
        <li>
            <input type="checkbox" name="task1" id="task1" />
            <label for="task1">Learn jQuery</label>
        </li>
        <li>
            <input type="checkbox" name="task2" id="task2" />
            <label for="task2">Learn Progressive Enhancement</label>
        </li>
        <li>
            <input type="checkbox" name="task3" id="task3" />
            <label for="task3">Build Great Websites</label>
        </li>
    </ul>
    <input type="submit" value="Mark Complete" />
</form>
</body>
</html>
```

The input form in this page doesn't require JavaScript. The user checks off the tasks he has completed and submits the form, and then it would be up to the server to load a new page with the completed tasks removed from the list.

Now, we can progressively enhance the page using jQuery: we bind an event handler to the checkboxes that mimics a standard form submit, by getting the submit URL for the form and generating POST data showing that the checkbox was checked. Then we animate the removal of the task to provide feedback to the user. We also hide the submit button because marking tasks complete has become an instantaneous process.

Discussion

Although few people browse without JavaScript these days, it's still a good practice when possible to build your pages so they work fine without JavaScript and then use jQuery and JavaScript to enhance them.



Beware that you don't make the user experience *worse* with JavaScript enhancements. The non-JavaScript version of this page may not give immediate feedback when you check off a task, but it does give you a way to change your mind easily if you make a mistake: either uncheck it before submitting or just don't submit the form at all.

If you "submit" each checkbox immediately when it's clicked, be sure you provide a way for your visitor to undo that action. If the task item disappears from the page, people will be afraid to click for fear of clicking the wrong item. You could either leave the item in the page but move it to a "completed" section or add an explicit Undo option.

5.21 Making Your Pages Accessible

Problem

You're building a web application with complex widgets and lots of Ajax functionality, but you want to accommodate visitors with disabilities.

Solution

Add keyboard accessibility and Accessible Rich Internet Applications (ARIA) semantics to your widgets. In the following code, the changes to support these features are indicated in **`bold`**:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <meta http-equiv="Content-Language" content="en-us" />
    <title>Dialog Demo</title>

    <style type="text/css">
        table {
            border-collapse: collapse;
            width: 500px;
        }
        th, td {
            border: 1px solid #000;
            padding: 2px 5px;
        }
        .dialog {
            position: absolute;
            background-color: #fff;
            border: 1px solid #000;
            width: 400px;
            padding: 10px;
        }
        .dialog h1 {
            margin: 0 0 10px;
        }
```

```
        .dialog .close {
            position: absolute;
            top: 10px;
            right: 10px;
        }
    </style>

    <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.js">
    </script>

    <script type="text/javascript">

        $(document).ready( function() {
            function close() {
                dialog.hide();
                $('#add-user').focus();
            }

            var title = $('<h1>Add User</h1>')
                    .attr('id', 'add-user-title'),

                closeButton = $('<button>close</button>')
                    .addClass('close')
                    .click(close)
                    .appendTo(title),

                content = $('<div/>')
                    .load('add.html'),

                dialog = $('<div/>')
                    .attr({
                        role: 'dialog',
                        'aria-labelledby': 'add-user-title'
                    })
                    .addClass('dialog')
                    .keypress(function(event) {
                        if (event.keyCode == 27) {
                            close();
                        }
                    })
                    .append(title)
                    .append(content)
                    .hide()
                    .appendTo('body');

            $('#add-user').click(function() {
                var height = dialog.height(),
                    width = dialog.width();

                dialog
                    .css({
                        top: ($(window).height() - height) / 2
                            + $(document).scrollTop(),
                        left: ($(window).width() - width) / 2
```

```
                        + $(document).scrollLeft()
                    })
                    .show();

                dialog.find('#username').focus();

                return false;
            });
        });

    </script>
</head>
<body>
<h1>Users</h1>
<a id="add-user" href="add.html">add a user</a>
<table>
<thead>
    <tr>
        <th>User</th>
        <th>First Name</th>
        <th>Last Name</th>
    </tr>
</thead>
<tbody>
    <tr>
        <td>jsmith</td>
        <td>John</td>
        <td>Smith</td>
    </tr>
    <tr>
        <td>mrobertson</td>
        <td>Mike</td>
        <td>Robertson</td>
    </tr>
    <tr>
        <td>arodriguez</td>
        <td>Angela</td>
        <td>Rodriguez</td>
    </tr>
    <tr>
        <td>lsamseil</td>
        <td>Lee</td>
        <td>Samseil</td>
    </tr>
    <tr>
        <td>lweick</td>
        <td>Lauren</td>
        <td>Weick</td>
    </tr>
</tbody>
</table>
</body>
</html>
```

We've added several useful features with just a small amount of additional code:

- We added ARIA semantics (`role` and `aria-labelledby`) so that assistive technology devices such as screen readers know that our `<div>` is a dialog and not just additional content on the page.
- We placed the keyboard focus in the dialog's first input field when it opens. This is helpful for all your visitors, sighted and nonsighted alike.
- We moved the keyboard focus back to the Add Users link when the dialog closes.
- We allowed the dialog to be canceled with the Escape key.

Discussion

ARIA is a work in progress, so browser and screen reader support for it is still limited. But by adding it now, you'll be better prepared for those visitors who can use it. And improved keyboard access benefits all your visitors.

For more information about ARIA, see the following:

- WAI-ARIA Overview
- DHTML Style Guide

Don't be thrown off by the old-school DHTML name; the DHTML Style Guide is an up-to-date keyboard accessibility reference for all the latest widgets.