jQuery Enlightenment

Cody Lindley

1st Edition, based on jQuery 1.3.2

Table of Contents

About the Author

Cody Lindley is a Christian, husband, son, father, brother, outdoor enthusiast, and client-side engineer. Since 1997 he has been passionate about HTML, CSS, JavaScript, Flash, Interaction Design, Interface Design, and HCI. He is best known in the jQuery community for creating Thickbox, a modal/dialog solution. In 2008 he officially joined the jQuery team as an evangelist. His current focus is on client-side optimization techniques as well as speaking and writing about jQuery. He is currently employed by Ning.

About the Reviewers

Paul Irish is a front-end developer, user-experience designer, and emerging interactions consultant at Molecular. He has created rich experiences for clients such as Reebok, Adidas, Boost Mobile, Finish Line, and Monster.com. He has written a few jQuery plugins, including idleTimer and Infinite Scroll. You can find him helping n00bz on the #jquery IRC channel or writing about JavaScript and jQuery at PaulIrish.com.

Jonathan Sharp is a standards driven freelance web designer and developer and founder of Out West Media LLC. With experience in both front-end and back-end technologies he brings value in integration delivering a seamless user experience. Jonathan has also developed a number of jQuery plugins such as jdMenu, jdNewsScroll and positionBy. Prior to freelancing, Jonathan worked for Union Pacific Railroad, CSC and Motorola, Inc. in Chicago after helping found Imprev, Inc. in Bellevue, WA in early 2000. He lives in Nebraska with his wife, Erin, and their daughter Noel. When not working he enjoys spending time with his family, playing with their dogs, and riding off into the sunset on Micah, his draft horse.

Nathan Smith has been building websites since late last century. He enjoys hand coding HTML, CSS, and JavaScript. He works as a UX developer at Fellowship Tech, and holds a Master of Divinity degree from Asbury Theological Seminary. He started Godbit, 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. Nathan blogs semi-regularly at his personal site SonSpring.

Jonathan Snook moves effortlessly from client-side, front-end work to hardcore server-side challenges, and his fluency in CSS, JavaScript, PHP and MySQL makes him the "turn-to" man for many high-profile clients. Co-author of Accelerated DOM Scripting and The Art and Science of CSS, he writes regularly at his popular blog Snook.ca, and for Sitepoint. Jonathan also works with his partners at Sidebar Creative, makers of world-class websites and innovative applications.

Jörn Zaefferer is a member of the core jQuery team. Along his work on jQuery itself, he wrote and maintains several of the most popular jQuery plugins. Jörn works as a consultant for Maxence Integration Technologies GmbH in Cologne, Germany, where he architects and develops Java-based web applications for Maxence's customers and Maxence's own products.

Introduction

jQuery Enlightenment was written to express, in short-order, the concepts essential to intermediate and advanced jQuery development. Its purpose is to instill in you, the reader, practices that jQuery developers take as common knowledge. Each chapter contains concepts essential to becoming a seasoned jQuery developer.

This book is intended for three types of readers. The first is someone who has read introductory books on jQuery and is looking for the next logical step. The second type of reader is a JavaScript developer, already versed in another library, now trying to quickly learn jQuery. The third reader is myself, the author. I crafted this book to be used as my own personal reference point for jQuery concepts. This is exactly the type of book I wish every JavaScript library had available.

Preface

Before you begin, it is important to understand the various stylistic nuances employed throughout this book. Please do not skip this section because it contains information that will aid you as you read.

jQuery semantics

The term "jQuery function" refers to the jQuery constructor function (`jQuery()` or alias `$()`) that is used to create an instance of the jQuery object.

The term "wrapper set" refers to DOM elements that are wrapped within jQuery methods. Specifically, this term is used to refer to the elements selected using the jQuery function. You may have heard this referred to as a "jQuery collection." In this book I will be using the term "wrapper set" instead of "jQuery collection."

How the book is structured

The book is organized into chapters loosely based on the arrangement of the jQuery API documentation. Each chapter contains isolated jQuery concepts relevant to the chapter's title. If you have not spent any time looking over the documentation on jquery.com I suggest you do so before reading this book.

More code, less words

This book is purposely written with the intention that the reader will examine the code examples closely. The text should be viewed as secondary to the code itself. It is my opinion that a code example is actually worth a thousand words. Do not worry if you initially find the explanations in the book to be confusing. Examine the code. Tinker with it. Re-read the code comments. Repeat this process until the material becomes clear. This is the level of expertise I hope you achieve, where documented code is all that is necessary for you to understand new development concepts.

Why Oh Why did I use `alert()` for code examples?

Believe me, I hate the `alert()` method as much as you do. But like it or not, it works reliably in every browser. To borrow a line from Dr. Seuss: It works "here, there, and everywhere!" It is not necessarily ideal, but I did not want the added complexity of `console` solutions to adversely affect code clarity. It is my goal to cut away any code overhead not directly supporting the concepts being taught.

Color coding

When grokking the code examples in this book the color **orange** is used to highlight code that you should examine closely or is the main point of the coded example. Any additional code used to support the focus of the coded examples will be colored **green**. The color **purple** is reserved for HTML and JavaScript comments.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<!-- HTML comment -->
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
// JavaScript comment
var focusOnThisCode = true;
</script>
</body>
</html>
```

Completely grok jQuery `text()` before reading this book

The code examples in this book make heavy use of the jQuery `text()` method. You need to be aware that the `text()` method, when used on a wrapper set containing more than one element, will actually combine and return a string of text contained in all elements of the wrapper set. This might be confusing if you were expecting it to return only the text in the first element of the wrapper set. Below is an example of how the `text()` method concatenates the strings found in the elements of a wrapper set.

JS Bin: http://jsbin.com/otona/edit/#html

```html
<!DOCTYPE html>
<html lang="en">
```

10

```
<body>
<span>I </span><span>love </span><span>jQuery</span><span>!</span>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
alert(jQuery('span').text()); // Alerts "I love jQuery!"
</script>
</body>
</html>
```

What is JS Bin and why do code examples use it?

JS Bin is a web application specifically designed to help JavaScript and CSS developers test snippets of code.

Since this book relies so heavily on code examples to express jQuery concepts, I thought it critical that the examples be available for immediate browser rendering and tinkering. JS Bin provides this functionality.

Having the code examples available to you immediately provides the means to manipulate and tinker with the code from any web browser. I not only encourage you to do this but have authored this book counting on the fact that you will need to tinker with the code while you are reading and learning.

In the top right corner of each coding block you will find a link to the HTML tab of a JS Bin code snippet on jsbin.com. From there you can edit the code and test your results in real time. This is a feature I have always wanted in a technical book, and have taken this opportunity to add it to jQuery Enlightenment.

Chapter 1 - Core jQuery

Base concept behind jQuery

While there are some conceptual variations (e.g. functions like **`$.ajax`**) in the jQuery API, the central concept behind jQuery is, "find something, do something." More specifically, select DOM element(s) from an HTML document and then do something with them using jQuery methods. This is the big picture concept.

To drive this concept home, reflect upon the code below.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<!-- jQuery will change this -->
<a href=""></a>
<!-- to this
<a href="http://www.jquery.com">jQuery</a>
-->
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
jQuery('a').text('jQuery').attr('href', 'http://www.jquery.com');
</script>
</body>
</html>
```

Notice that in this HTML document we are using jQuery to select a DOM element (**`<a>`**). With something selected, we then do something with the selection by invoking the jQuery methods **`text()`** and **`attr()`**. Grokking the "find something, do something" foundational concept is critical to advancing as a jQuery developer.

The concept, behind the concept, behind jQuery

While selecting something and doing something is the core concept behind jQuery, I would like to extend this concept to include creating something as well. Therefore, the concept behind jQuery could be extended to include first creating something new, selecting it, then doing something with it. We could call this the concept, behind the concept, behind jQuery.

What I am trying to make obvious is that you are not stuck with only selecting something that is already in the DOM. It is additionally important to grok that jQuery can be used to create new DOM elements and then do something with these elements.

In the code example below we create a new `<a>` element that is not in the DOM. Once created, it is selected and then manipulated.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
jQuery('<a>jQuery</a>').attr('href', 'http://www.jquery.com').appendTo('body');
</script>
</body>
</html>
```

The key concept to pickup on here is that we are creating the `<a>` element on the fly and then operating on it as if it was already in the DOM.

How to check the current jQuery version

Scenarios may exist where working with the latest and greatest version of jQuery is not possible. In these situations it is extremely helpful to know which version of jQuery you are dealing with. Hopefully it is documented in the file itself, or the URL that includes the jQuery source, but if not we can extract the information from jQuery itself. Below are two solutions for checking the version of jQuery in use.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
alert(jQuery.fn.jquery);
alert(jQuery().jquery);
</script>
</body>
</html>
```

jQuery requires HTML run in *standards* mode or *almost standards* mode

There are known issues with jQuery methods not working correctly when a browser renders an HTML page in quirks mode. Make sure when you are using jQuery that the browser interprets the HTML in *standards* mode or *almost standards* mode by using a [valid doctype](#).

To ensure proper functionality, code examples in this book use the HTML 5 doctype.

```html
<!DOCTYPE html>
```

Include all CSS files before including jQuery

As of jQuery 1.3, the library no longer guarantees that all CSS files are loaded before it fires the custom `ready()` event. Because of this change in jQuery 1.3, you should always include all CSS files before any jQuery code. This will ensure that the browser has parsed the CSS before it moves on to the JavaScript included later in the HTML document. Of course, images that are referenced via CSS may or may not be downloaded as the browser parses the JavaScript.

Using a hosted version of jQuery

When embedding jQuery into a web page most people choose to download the [source code](#) and link to it from a personal domain/host. However, there are other options that involve someone else hosting the jQuery code for you.

[Google hosts](#) several versions of the jQuery source code with the intent of it being used by anyone. This is actually very handy. In the code example below I am using a `<script>` element to include a minified version of jQuery that is hosted by Google.

JS Bin: http://jsbin.com/ivifo/edit/#html

```html
<!DOCTYPE html>
<html lang="en">
<body>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>alert('It is ' + ('jQuery' in window) + ' jQuery is loaded');</script>
</body>
</html>
```

Google also hosts several previous versions of the jQuery source code, and for each version, a minified and non-minified variant is provided. I recommend using the non-minified variant during

development, as debugging errors is always easier when you are dealing with non-minifed code.

A benefit of using a Google hosted version of jQuery is that it is reliable, fast, and potentially cached.

Notes:

- You can actually load the most recent version (or a precise version) of jQuery by removing version fields. As an example this functionality makes it possible to always load the most recent version of 1.3.X. When the library release 1.3.9 you don't have to update the link to google, google will actually serve the most recent version of 1.3.X.

- When directly linking to the library you only get 1 day of caching, the `google.load()` technique uses more aggressive caching.

- jQuery UI is also being hosted by google for public consumption.

Executing code when the DOM is ready, but before `window.onload`

The `window.onload` native JavaScript event can be used to programmatically handle when a web page has been completely loaded in the browser. However, because the event indicating everything has loaded does not fire until all assets (images, Flash, etc) are loaded, waiting for this event can be rather time consuming. A better solution would be to start scripting the DOM as soon as it is available to be manipulated.

jQuery provides a custom event that is executed after the DOM is ready to be scripted but before the `window.onload` event fires. This is a custom jQuery event called `ready()`. There is one argument passed to this function which is a reference to the jQuery function.

Below I give three coded examples of this custom event in use.

JS Bin: http://jsbin.com/iciku/edit/#html

```html
<!DOCTYPE html>
<html lang="en">
<head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>

// Standard
jQuery(document).ready(function(){
    alert('DOM is ready!');
});

// Shortcut, but same thing as above
jQuery(function(){
    alert('No really, the DOM is ready!');
});
```

```
// Shortcut with fail-safe usage of $. Keep in mind that a reference
// to the jQuery function is passed into the anonymous function.
jQuery(function($){
    alert('Seriously its ready!');
// Use $() with out fear of conflicts
});

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

Notes:

- Keep in mind that you can attach as many `ready()` events to the document as you would like. You are not limited to only one. They are executed in the order they were added.

- Make sure all stylesheets are included before your jQuery code. Doing so will make sure all the elements in the DOM are correctly rendered before jQuery begins executing code.

Executing jQuery code when the browser window is completely loaded

Typically we do not want to wait for the `window.onload` event. That is the point of using a custom event like `ready()` that will execute code before window loads, but after the DOM is ready to be traversed and manipulated.

However, sometimes we actually do want to wait. While the custom `ready()` event is great for executing code once the DOM is available, we can also use jQuery to execute code once the entire web page (including all assets) is completely loaded.

This can be done by attaching a load event handler to the `window` object. jQuery provides the `load()` event method that can be used to invoke a function once the window is completely loaded. Below I provide an example of the `load()` event method in use.

JS Bin: http://jsbin.com/elawe/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>

// Pass window to the jQuery function and attach
// event handler using the load() method shortcut.
jQuery(window).load(function(){
```

```
      alert('The page and all its assets are loaded!');
});


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

Execute jQuery code when DOM is parsed, without using `ready()`

The custom `ready()` event is not entirely needed. If your JavaScript code does not affect the DOM, you can include it anywhere in the HTML document. This means you can avoid the `ready()` event altogether if your JavaScript code is not dependent on the DOM being intact.

Most JavaScript nowadays, especially jQuery code, will involve manipulating the DOM. This means the DOM has to be fully parsed by the browser in order for you to operate on it. This fact is why developers have been stuck on the `window.onload` roller coaster ride for a couple of years now.

To avoid using the `ready()` event for code that operates on the DOM you can simply place your code in an HTML document before the closing `</body>` element . Doing so ensures the DOM is completely loaded, simply because the browser will parse the document from top to bottom. If you place your JavaScript code in the document after the DOM elements it manipulates, there is no need to use the `ready()` event.

In the example below I have forgone the use of `ready()` by simply placing my script before the document body closes. This is the technique I use throughout this book and on the majority of sites that I build.

JS Bin: http://jsbin.com/apeha/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<p>Hi, I'm the DOM! Script away!</p>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
alert(jQuery('p').text());
</script>
</body>
</html>
```

If I was to place the `<script>` before the `<p>` element, it would execute before the browser had loaded the `<p>` element. This would cause jQuery to assume the document does not contain any `<p>` elements. However, if I were to use the jQuery custom `ready()` event then jQuery would not execute the code until the DOM was ready. But why do this, when we have control over the location of the `<script>` element in the document? Placing jQuery code at the bottom of the page avoids having to using the `ready()` event. In fact, placing all JavaScript code at the bottom of a page is a proven performance strategy.

Use the $ alias without fear of conflicts

jQuery uses the $ character as an alias for the jQuery namespace, but our love affair with the $ character is not exclusive, because other libraries have also jumped on the $ bandwagon. But wait! You do not have to end the love affair. You can just get your own wagon.

You can still use $ by passing a self executing anonymous function to the jQuery namespace. Passing this function, the `jQuery` object/namespace allows you to use the $ character as a parameter within the function, thus creating a unique scope (a.k.a - a closure). Translation: You now have a private scope in which you can freely use $ without fear of conflicting with another JavaScript library that might be running on the same HTML page. Below, I provide a working HTML page as an example of this concept.

JS Bin: http://jsbin.com/ewamu/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){
// Use $ alias worry-free of conflicts
alert('You are using jQuery ' + $().jquery );
})(jQuery)
</script>
</body>
</html>
```

Grokking jQuery chaining

Once you have selected something using the jQuery function and created a wrapper set, you can actually chain jQuery methods to the DOM elements contained inside of the set. Conceptually, jQuery methods continue the chain by always returning the jQuery wrapper set. It can then be used by the next jQuery method in the chain. Note: Most jQuery methods are chainable, but not all.

You should always attempt to re-use the wrapped set by leveraging chaining. In the code below the `text()`, `attr()`, and `addClass()` method are being chained.

JS Bin: http://jsbin.com/ojodu/edit/#html

```html
<!DOCTYPE html>
<html lang="en">
<body>
<a></a>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){
 $('a')
    .text('jQuery') // Returns $('a')
    .attr('href', 'http://www.jquery.com/') // Returns $('a')
    .addClass('jQuery'); // Returns $('a')
})(jQuery)
</script>
</body>
</html>
```

Breaking the chain with destructive methods

As mentioned before, not all jQuery methods maintain the chain. There are methods like `text()` that can be chained when used to *set* the text node of an element. However, `text()` actually breaks the chain when used to *get* the text node contained within an element.

In the example below `text()` is used to *set* and then *get* the text contained with the `<a>` element.

JS Bin: http://jsbin.com/uqeqi/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<p></p>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){
 var theText = $('p')
    .text('jQuery')
    .text(); // Returns the string "jQuery"

alert(theText);
// Cannot chain after text(). The chain is broken.
// A string is returned, not the jQuery object.

})(jQuery)
</script>
</body>
</html>
```

Getting the text contained within an element using `text()` is a prime example of a broken chain because the method will return a string containing the text node, but not the jQuery wrapper set.

It should be no surprise that if a jQuery method does not return the jQuery wrapper set, the chain is thereby broken. This method is considered to be destructive.

Using destructive jQuery methods and exiting destruction using `end()`

jQuery methods that alter the original jQuery wrapper set selected are considered to be destructive. The reason being that they do not maintain the original state of the wrapper set. Not to worry; nothing is really destroyed or removed. Rather, the former wrapper set is attached to a new set.

However, fun with chaining does not have to stop once the original wrapped set is altered. Using the `end()` method you can back out of any destructive changes made to the original wrapper set. Examine the usage of the `end()` method in the following example to understand how to operate *in* and *out* of DOM elements.

JS Bin: http://jsbin.com/iluhi/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<style>
.last {background: #900}
</style>
<ul id="list">
    <li></li>
```

```
    <li>
        <ul>
            <li></li>
            <li></li>
            <li></li>
        </ul>
    </li>
    <li></li>
    <li></li>
</ul>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

$('#list') // Original wrapper set
    .find('> li') // Destructive method
        .filter(':last') // Destructive method
            .addClass('last')
        .end() // End .filter(':last')
        .find('ul') // Destructive method
            .css('background', '#ccc')
                .find('li:last') // Destructive method
                    .addClass('last')
                .end() // End .find('li:last')
        .end() // End .find('ul')
    .end() // End .find('> li')
.find('li') // Back to the orginal $('#list')
    .append('I am an &lt;li&gt;');

})(jQuery);
</script>
</body>
</html>
```

The jQuery function is multifaceted

The jQuery function is multifaceted. We can pass it differing values and string formations that it can then use to perform unique functions. Here are several uses of the jQuery function:

- Select elements from the DOM using CSS expressions & custom jQuery expressions. As well as, selecting elements using DOM references: `jQuery('p > a')` or `jQuery(':first')` and `jQuery(document.body)`
- Create HTML on the fly by passing HTML string structures or DOM methods that create DOM elements: `jQuery('<div id="nav"></div>')` or `jQuery(document.createElement('div'))`
- A shortcut for the `ready()` event by passing a function to the jQuery function: `jQuery(function($){ /* Shortcut for ready() */ })`

21

Each of these usages are detailed in the code example below.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
jQuery(function($){ // Pass jQuery a function

// Pass jQuery a string of HTML
$('<p></p>').appendTo('body');

// Pass jQuery an element reference
$(document.createElement('a')).text('jQuery').appendTo('p');

// Pass jQuery a CSS expression
$('a:first').attr('href', 'http://www.jquery.com');

// Pass jQuery DOM reference
$(document.anchors[0]).attr('jQuery');

});
</script>
</body>
</html>
```

Though not common practice, it is also possible to pass the jQuery function an array of elements.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<ul>
    <li>dog</li>
    <li>cat</li>
</ul>
<script type="text/JavaScript" src="http://code.jquery.com/jquery-latest.js"></script>
<script>
var liElements = jQuery('ul li').get(); // Get array of li elements. Ends the chain.
alert(jQuery(liElements).eq(0).text()); // Alerts "dog"
</script>
</body>
</html>
```

Grokking when the keyword `this` refers to DOM elements

When attaching events to DOM elements contained in a wrapper set, the keyword `this` can be used to refer to the current DOM element invoking the event. The following example contains jQuery code that will attach a custom `mouseenter` event to each `<a>` element in the page. The native JavaScript `mouseover` event fires when the cursor enters or exits a child element, whereas jQuery's `mouseenter` does not.

JS Bin: http://jsbin.com/akuce/edit/#html

```html
<!DOCTYPE html>
<html lang="en">
<body>
<a id="link1">jQuery.com</a>
<a id="link2">jQuery.com</a>
<a id="link3">jQuery.com</a>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){
    $('a').mouseenter(function(){
        alert(this.id);
    });
})(jQuery);
</script>
</body>
</html>
```

Inside the anonymous function that is passed to the `mouseenter()` method, we use the keyword `this` to refer to the current `<a>` element. Each time the mouse touches the "jQuery.com" text, the browser will alert which element has been moused-over by identifying its `id` attribute value.

In the previous example it is also possible to take the `this` reference and pass it to the jQuery function so that the DOM element is wrapped with jQuery functionality.

So instead of this:

```javascript
// Access the ID attribute of the DOM element.
alert(this.id);
```

We could have done this:

```javascript
// Wrap the DOM element with jQuery object,
// then use attr() to access ID value.
alert($(this).attr('id'));
```

This is possible because the jQuery function not only takes selector expressions, it will also take references to DOM elements. In the code `this` is a reference to a DOM element.

The reason that you might want to wrap jQuery functionality around a DOM element should be obvious. Doing so gives you the ability to now use jQuery chaining, should you have need for it.

Iterating over a set of elements contained in the jQuery wrapper set is somewhat similar to the concept we just discussed. By using the jQuery `each()` method we can iterate over each DOM element contained in a wrapper set. This allows access to each DOM element individually, via the usage of the `this` keyword.

Building upon the markup in the previous example, we can select all `<a>` in the page and use the `each()` method to iterate over each `<a>` element in the wrapper set, accessing its `id` attribute. Here is an example.

JS Bin: http://jsbin.com/aqeki/edit/#html

```html
<!DOCTYPE html>
<html lang="en">
<body>
<a id="link1">jQuery.com</a>
<a id="link2">jQuery.com</a>
<a id="link3">jQuery.com</a>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){
    $('a').each(function(){ // Loop that alerts the id value for every <a> in the page
        alert($(this).attr('id')); // "this" refers to the current element in the loop
    });
})(jQuery);
</script>
</body>
</html>
```

If you were to load the HTML in a browser, the browser would alert for you the `id` value of each `<a>` element in the page. Since there are three `<a>` elements in the page you get three iterations via the `each()` method and three alert windows.

Extract elements from a wrapper set use them directly without jQuery

Just because you wrap jQuery functionality around an HTML element does not mean that you lose access to the actual DOM element itself. You can always extract an element from the wrapper set and operate on the element via native JavaScript. For example, in the code below I am setting the title attribute of the `<a>` element in the HTML page by manipulating the native title property of the `<a>` DOM element.

```
                                               JS Bin: http://jsbin.com/imeqo/edit/#html

<!DOCTYPE html>
<html lang="en">
<body>
<a>jQuery.com</a>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>

(function($){
    // Using DOM node properties to set the title attribute
    $('a').get(0).title = 'jQuery.com';

    // Manipulation of DOM element using jQuery methods
    $('a').attr('href', 'http://www.jquery.com');
})(jQuery);

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

As demonstrated, jQuery provides the handy `get()` method for accessing DOM elements at a specific index in the wrapper set.

But there is another option here. You can avoid using the `get()` method by simply using the square bracket array notation on the jQuery object itself. In the context of our prior code example:

This code:

```
$('a').get(0).title = 'jQuery.com';
```

Could become this:

```
$('a')[0].title = 'jQuery.com';
```

Both allow access to the actual DOM element. Personally, I prefer square bracket notation. It is faster because it uses native JavaScript to retrieve the element from an array, instead of passing it to a method.

However, the `get()` method provides a slick solution for placing all of the DOM elements into a native Array. By simply calling the `get()` method without passing it an index parameter the method will return all of the DOM elements in the wrapper set in a native JavaScript array.

To demonstrate, let's take `get()` for a test drive. In the code below I am placing all the `<a>` elements into an array. I then use the array to access the title property of the third `<a>` DOM object on the page.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<a href="http://www.jquery.com" title="anchor1">jQuery.com</a>
<a href="http://www.jquery.com" title="anchor2">jQuery.com</a>
<a href="http://www.jquery.com" title="anchor3">jQuery.com</a>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){
    var arrayOfAnchors = $('a').get(); // Create native array from wrapper set
    alert(arrayOfAnchors[2].title); // Alerts the third link
})(jQuery);
</script>
</body>
</html>
```

Notes:

- Using `get()` will end jQuery's chaining. It will take the wrapper set and change it into a simple array of DOM elements that are no longer wrapped with jQuery functionality. Therefore, using the `.end()` method cannot restore chaining after `.get()`.

Checking to see if the wrapper set is empty

Before you begin to operate on a wrapper set, it is logical to check that you have, in fact, selected something. The simplest solution is to use an `if` statement to check if the wrapper set contains any DOM elements.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<a>jQuery</a>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>

if (jQuery('a').get(0)){ // Is there an element in the set?
    jQuery('a').attr('href', 'http://www.jquery.com');
}

if (jQuery('a').length){ // Check the length of the set. Can also use .size()
    jQuery('a').attr('title', 'jQuery');
}

</script>
```

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

The truth of the matter is, the above `if` statements are not totally necessary, because jQuery will fail silently if no elements are found. However, each method chained to any empty wrapper set still gets invoked. So while we could actually forgo the use of the `if` statements, it is likely a good rule of thumb to use them. Invoking methods on an empty wrapper set could potentially cause unnecessary processing, as well as undesirable results if methods return values other than the wrapper set, and those values are acted upon.

Creating an alias by renaming the jQuery object itself

jQuery provides the `noConflict()` method which has several uses. Namely, the ability to replace `$` with another alias. This can be helpful in three ways. It can relinquish the use of the `$` sign to another library, help avoid potential conflicts, and provide the ability to customize the namespace/ alias for the jQuery object.

For example, let's say that you are building a web application for company XYZ. It might be nice to customize jQuery so that instead of having to use `jQuery('div').show()` or `$('div').show()` you could use `XYZ('div').show()` instead.

JS Bin: http://jsbin.com/egaga/edit#html

```html
<!DOCTYPE html>
<html lang="en">
<body>
<div>XYZ company</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
var XYZ = jQuery.noConflict();
// Do something with jQuery methods
alert(XYZ("div").text());
</script>
</body>
</html>
```

Notes:

- By passing the `noConflict()` function a boolean value of true you can completely undo what jQuery has introduced into the web page. This should only be used in extreme cases because it will, more than likely, cause issues with jQuery plugins.

Using `.each()` when implicit iteration is not enough

Hopefully it is obvious that if you have an HTML page (example below) with three empty **<div>** elements, the following jQuery statement will select all three elements on the page, iterate over the three elements (implicit iteration) and will insert the text "I am a div" in all three **<div>** elements.

```
<!DOCTYPE html>
<html lang="en">
<body>
<div></div>
<div></div>
<div></div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){
$('div').text('I am a div');
})(jQuery);
</script>
</body>
</html>
```

This is considered implicit iteration because jQuery code assumes you would like to manipulate all three elements, which requires iterating over the elements selected and setting the text node value of each **<div>** with the text "I am a div". When this is done by default, it is referred to as implicit iteration.

This is pretty handy. For the most part, the majority of the jQuery methods will apply implicit iteration. However, other methods will only apply to the first element in the wrapper set. For example, the jQuery method **attr()** will only access the first element in the wrapper set when used to get an attribute value.

> **Notes:**
>
> - When using the **attr()** method to *set* an attribute, jQuery will actually apply implicit iteration to set the attribute and its value to *all* the elements in the wrapper set.

In the code below, the wrapper set contains all **<div>** elements in the page, but the **attr()** method only returns the **id** value of the first element contained in the wrapper set.

```
<!DOCTYPE html>
<html lang="en">
<body>
<div id="div1">I am a div</div>
<div id="div2">I am a div</div>
```

```
<div id="div3">I am a div</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){
// Alerts attribute value for first element in wrapper set
  alert($('div').attr('id')); // Alerts "div1"
})(jQuery);
</script>
</body>
</html>
```

For the sake of demonstration, assume your goal is actually to get the id attribute value for each element on the page. You could write three seperate jQuery statements accessing each `<div>` element's `id` attribute value. If we were to do that it might look something like this:

```
$('#div1').attr('id');
$('#div2').attr('id');
$('#div3').attr('id');

// or

var $divs = $('div'); // Cached query
$divs.eq(0).attr('id'); // Start with 0 instead of 1
$divs.eq(1).attr('id');
$divs.eq(2).attr('id');
```

That seems a bit verbose, no? Wouldn't it be nice if we could loop over the wrapper set and simply extract the `id` attribute value from each of the `<div>` elements? By using the `$().each()` method we invoke another round of iteration when our wrapper set requires explicit iteration to handle multiple elements.

In the code example below I use the `$().each()` method to loop over the wrapper set, access each element in the set, and then extract its `id` attribute value.

```
<!DOCTYPE html>
<html lang="en">
<body>
<div id="div1">div1</div>
<div id="div2">div2</div>
<div id="div3">div3</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){
    // 3 alerts, one for each div
  $('div').each(function(){
        // "this" is each element in the wrapper set
        alert($(this).attr('id'));
        // Could also be written: alert(this.id);
    });
```

```
})(jQuery);
</script>
</body>
</html>
```

Imagine the possibilites ahead of you with the ability to enforce iteration, any time you please.

> **Notes:**
>
> - jQuery also provides a `$.each` function, not to be confused the `$().each` method which is used specifically to iterate over a jQuery wrapper set. The `$.each` can actually be used to iterate over any old JavaScript array or object. It is essentailly a substitue for native JavaScript loops.

Elements in jQuery wrapper set returned in document order

As of jQuery 1.3.2, the selector engine will return results in document order as opposed to the order in which the selectors were passed in. The wrapper set will be populated with the selected elements based on the order each element appears in the document, from top to bottom. Before 1.3.2, this was not the case.

JS Bin: http://jsbin.com/ebuhe/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<h1>h1</h1>
<h2>h2</h2>
<h3>h3</h3>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){
    // We pass in h3 first, but h1 appears earlier in
    // the document, so it is first in the wrapper set.
    alert($('h3, h2, h1').get(0).nodeName); // Alerts "H1"
})(jQuery);
</script>
</body>
</html>
```

Determining context used by the jQuery function

The default context used by the jQuery function when selecting DOM elements is the document element (e.g. `$('a', document)`). This means that if you do not provide the jQuery function (e.g.

`jQuery()`) with a second parameter to be used as the context for the DOM query the default context used is the document element, more commonly known as `<body>`.

It is possible to determine the context in which the jQuery function is performing a DOM query by using the `context` property. Below I show two coded examples of retrieving the value of the context property.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<div>
    <div>
        <div id="context">
            <a href="#">jQuery</a>
        </div>
    </div>
</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($) {
    // Alerts "object HTMLDocument" in Firefox
// Same as $('a', document).context;
    alert($('a').context);

// Alerts "object HTMLDivElement" in FireFox
    alert($('a', $('#context')[0]).context);
})(jQuery);
</script>
</body>
</html>
```

Create an entire DOM structure, including DOM events, in a single chain

By leveraging chaining and jQuery methods we can create not only a single DOM element, but entire DOM structures. Below I create an un-ordered list of jQuery links that I then add to the DOM.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

jQuery('<ul></ul>')
    .append('<li><a>jQuery.com</a></li><li><a>jQuery Documentation</a></li>')
```

```
        .find('a:first')
        .attr('href', 'http://www.jquery.com')
    .end()
    .find('a:eq(1)')
        .attr('href', 'http://docs.jquery.com')
    .end()
    .find('a')
        .click(function(){
        return confirm('Leave this page?');
        })
    .end()
.appendTo('body');

})(jQuery);
</script>
</body>
</html>
```

The concept you need to take away from this example is jQuery can be used to craft and operate on complex DOM structures. Using jQuery methods alone, you can whip up most any DOM structure you might need.

Custom jQuery filters can select elements, when used alone

It is not necessary to provide an actual element in conjunction with a filter, such as
`$('div:hidden')`. It is possible to simply pass the filter alone, anywhere a selector expression is
expected.

Some examples:

```
// Selects all hidden elements
$(':hidden');

// Selects all div elements, then selects only even elements
$('div').filter(':even');
```

Grokking the `:hidden` and `:visible` filter

The custom jQuery selector filters `:hidden` and `:visible` do not take into account CSS visibility
property as one might expect. The way jQuery determines if an element is hidden or visible is if the
element consumes any space in the document. To be exact, an element is visible if its browser-
reported `offsetWidth` or `offsetHeight` is greater than 0. That way, an element that might have
a CSS `display` value of `block` contained in an element with a `display` value of `none` would
accurately report that it is not visible.

Examine the code carefully and make sure you understand why the value returned is `true` even
though the `<div>` being selected has an inline style of `display:block`.

JS Bin: http://jsbin.com/aqilu/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<div id="parentDiv" style="display:none;">
 <div id="childDiv" style="display:block;"></div>
</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
```

```
(function($){
    // Returns true because the parent div is hidden, so the
    // encapsulated div reports zero offsetWidth, offsetHeight
    alert($('#childDiv').is(':hidden'));
})(jQuery);
</script>
</body>
</html>
```

Using the `is()` method to return a Boolean value

It is often necessary to determine if the selected set of elements does, in fact, contain a specific element. Using the `is()` method, we can check the current set against an expression/filter. The check will return `true` if the set contains at least one element that is selected by the given expression/filter. If it does not contain the element, a `false` value is returned. Examine the following code:

```
<!DOCTYPE html>
<html lang="en">
<body>
<div id="i0">jQuery</div>
<div id="i1">jQuery</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){
    // Returns true
    alert($('div').is('#i1'));

    // Returns false. Wrapper set contains no <div> with id="i2"
    alert($('div').is('#i2'));

    // Returns false. Wrapper set contains no hidden <div>
    alert($('div').is(':hidden'));
})(jQuery);
</script>
</body>
</html>
```

It should be apparent that the second `alert()` will return a value of false because our wrapper set did not contain a `<div>` that had an `id` attribute value of `i2`. The `is()` method is quite handy for determining if the wrapper set contains a specific element.

Notes:

34

- As of jQuery 1.3 the `is()` method supports all expressions. Previously complex expressions, such as those containing hierarchy selectors (such as `+`, `~`, and `>`), always returned `true`.

- Filter is used by other internal jQuery functions. Therefore, all rules that apply there, apply here, as well.

- Some developers use `is('.class')` to determine if an element has a specific class, don't forget that jQuery already has a method for doing this called `hasClass('class')` which can be used on elements that contain more than one class value. But truth be told, `hasClass()` is just a convenient wrapper for the `is()` method.

You can pass jQuery more than one selector expression

You can provide the jQuery function's first parameter several expressions separated by a comma: `$('expression, expression, expression')`. In other words, you are not limited to selecting elements using only a single expression. For example, in the example below, I am passing the jQuery function three expressions separated by a comma.

```
                                                    JS Bin: http://jsbin.com/ewune/edit/#html
<!DOCTYPE html>
<html lang="en">
<body>
<div>jQuery </div>
<p>is the </p>
<ul>
    <li>best!</li>
</ul>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){
    // Alerts jQuery is the best!
    alert($('div, p, ul li').text());

    // Inefficient way. Alerts jQuery is the best!
    alert($('div').text() + $('p').text() + $('ul li').text());
})(jQuery);
</script>
</body>
</html>
```

Each of these expressions selects DOM elements that are all added to the wrapper set. We can then operate on these elements using jQuery methods. Keep in mind that all the selected elements will be placed in the same wrapper set. An inefficient way to do this would be to call the jQuery function three times, once for each expression.

Determining anything is selected by checking wrapper set `.length`

It is possible to determine if your expression has selected anything by checking if the wrapper set has a length. You can do so by using the array property, `length`. If the `length` property does not return 0, then you know at least one element matches the expression you passed to the jQuery function. For example, in the code below we check the page for an element with an `id` of "notHere". Guess what? It is not there!

```html
<!DOCTYPE html>
<html lang="en">
<body>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){
    // Alerts "0"
    alert($('#notHere').length);
})(jQuery);
</script>
</body>
</html>
```

Notes:

 - If it is not obvious the length property can also report the number of elements in the wrapper set. Stated another way, how many elements were selected by the expression passed to the jQuery function.

Create your own custom filters for selecting elements

It is possible to extend the capabilities of the jQuery selector engine by creating your own custom filters. In theory, all we are doing here is building upon the custom selectors that are already part of jQuery. For example, say we would like to select all elements on a web page that are absolutely positioned. Since jQuery does not already have a custom `:positionAbsolute` filter, we can create our own.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<div style="position:absolute">absolute</div>
<span style="position:absolute">absolute</span>
<div>static</div>
<div style="position:absolute">absolute</div>
<div>static</div>
```

```
<span style="position:absolute">absolute</span>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){
    // Define custom filter by extending $.expr[':']
    $.expr[':'].positionAbsolute = function(element) {
        return $(element).css('position') === 'absolute';
    };

    // How many elements in the page are absolutely positioned?
    alert($(':positionAbsolute').length); // Alerts "4"

    // How many div elements are absolutely positioned?
    alert($('div:positionAbsolute').length); // Alerts "2"
})(jQuery);
</script>
</body>
</html>
```

The most important thing to grasp here is that you are not limited to the default selectors provided by jQuery. You can create your own. However, before you spend the time creating your own version of a selector, you might just simply try the `filter()` method with a specified filtering function. For example, I could have avoided writing the `:positionAbsolute` selector by simply filtering the `<div>` elements in my prior example with a function I pass to the `filter()` method.

```
// Remove <div> elements from the wrapper
// set that are not absolutely positioned
$('div').filter(function(){return $(this).css('position') === 'absolute';});

// or

// Remove all elements from the wrapper
// set that are not absolutely positioned
$('*').filter(function(){return $(this).css('position') === 'absolute';});
```

Notes:

- For additional information about creating your own selectors I suggest the following read:
http://www.bennadel.com/blog/1457-How-To-Build-A-Custom-jQuery-Selector.htm

Differences between filtering by numeric order vs. DOM relationships

jQuery provides filters for filtering the wrapper set by the elements numerical context within the set.

These filters are:

- :first
- :last
- :even
- :odd
- :eq(index)
- :gt(index)
- :lt(index)

Using `:first` will select the first element in the set while `:last` will select the last element in the set. It is important to remember that they filter the set based on the relationship (numerical hierarchy starting at 0) within the set, but not the elements' relationships in the context of the DOM. Given this knowledge, it should be obvious why the filters `:first`, `:last`, `:eq(index)` will always return a single element.

If it is not obvious, allow me to explain further. The reason that `:first` can only return a single element is because there can only be one element in a set that is considered first when there is only one set. This should be fairly logical. Examine the code below to see this concept in action.

JS Bin: http://jsbin.com/ejoti/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<ul>
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
    <li>5</li>
</ul>
<ul>
    <li>6</li>
    <li>7</li>
    <li>8</li>
    <li>9</li>
    <li>10</li>
</ul>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){
```

```
// Remember that text() combines the contents of all
// elements in the wrapper set into a single string.
alert('there are ' + $('li').length + ' elements in the set');

// Get me the first element in the set
alert($('li:first').text()); // Alerts "1"

// Get me the last element in the set
alert($('li:last').text()); // Alerts "10"

// Get me the 6th element in the set, 0 index
alert($('li:eq(5)').text()); // Alerts "6"

})(jQuery);
</script>
</body>
</html>
```

With a clear understanding of manipulating the set itself we can augment our understanding of selecting elements by using filters that select elements that have unique relationships with other elements within the actual DOM. jQuery provides serveral selectors to do this. Some of these selectors are custom, while some are well known CSS expressions for selecting DOM elements.

- ancestor descendant
- parent > child
- prev + next
- prev ~ siblings
- :nth-child(selector)
- :first-child
- :last-child
- :only-child
- :empty
- :has(selector)
- :parent

Usage of these selector filters will select elements based on the elements relationship within the DOM as it pertains to other elements in the DOM. To demonstrate this concept let's look at some code.

```
<!DOCTYPE html>
<html lang="en">
<body>
<ul>
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
    <li>5</li>
</ul>
<ul>
```

```
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
    <li>5</li>
</ul>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

// Remember that text() combines the contents of all
// elements in the wrapper set into a single string.
alert($('li:nth-child(2)').text()); // Alerts "22"
alert($('li:nth-child(odd)').text()); // Alerts "135135"
alert($('li:nth-child(even)').text()); // Alerts "2424"
alert($('li:nth-child(2n)').text()); // Alerts "2424"

})(jQuery);
</script>
</body>
</html>
```

If you are surprised by the fact that `$('li:nth-child(odd)').text()` returns the value 135135 you are not yet grokking releationship filters. The statement, `$('li:nth-child(odd)')` said verbally would be, find all `<li>` elements in the web page that are children then filter them by odd children. Well, it just so happens that there are two structures in the page that have a grouping of siblings made up of `<li>`'s. My point is this: The wrapper set is made up of elements based on a filter that takes into account an element's relationship to other element in the DOM. These relationships can be found in multiple locations.

The concept to take-away here is that not all filters are created equally. Make sure you understand which ones filter based on DOM relationships — e.g. `:only-child` — and which ones filter by the elements position — e.g. `:eq()` — in the wrapped set.

Selecting elements by `id` when the value contains meta-characters

jQuery selectors use a set of meta-characters (e.g. `# ~ [] = >`) that when used as a literal part of a name (e.g. `id="#foo[bar]"`) should be escaped. It is possible to escape characters by placing two backslashes before the character. Examine the code below to see how using two backslashes in the selection expression allows us to select an element with an id attribute value of `#foo[bar]`.

JS Bin: http://jsbin.com/ohaqe/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
```

```
<div id="#foo[bar]">jQuery</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){
    // Alerts "jQuery"
    alert($('#\\#foo\\[bar\\]').text());
})(jQuery);
</script>
</body>
</html>
```

Here is the complete list of Characters that need to be escaped when used as a literal part of a name.

- #
- ;
- &
- ,
- .
- +
- *
- ~
- '
- :
- "
- !
- ^
- $
- [
-]
- (
-)
- =
- >
- |
- /

Stacking selector filters

It is possible to stack selector filters —
e.g. `a[title="jQuery"][href^="http://"][class!=""]`. The obvious example of this is selecting an element that has specific attributes with specific attribute values. For example, the jQuery code below will only select `<a>` elements in the HTML page that:

- Contain an href attribute with a starting value of "http://"

41

- Have a title attribute with a value of "jQuery"
- Do not have a class attribute

Only one **<a>** is being selected.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<a title="jQuery">jQuery.com</a>
<a href="http://www.jquery.com" title="jQuery" class="foo">jQuery.com</div>
<a href="">jQuery.com</a>
<a href="http://www.jquery.com" title="jQuery">jQuery.com</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){
    // Alerts "1"
    alert( $('a[title="jQuery"][href^="http://"][class!=""]').length);
})(jQuery);
</script>
</body>
</html>
```

Notice in the code how we have stacked three filters to accomplish this selection.

It is also possible to stack other selector filters besides just attribute filters. For example:

```javascript
// Select the last <div> contained in the
// wrapper set that contains text "jQuery"
$('div:last:contains("jQuery")')

// Get all checkboxes that are both visible and checked
$(':checkbox:visible:checked')
```

The take-away concept here is that selector filters can be stacked and used in combination.

Notes:

- You can also nest and stack filters — e.g. `$('p').filter(':not(:first):not(:last)')`

Nesting selector filters

It is possible to nest selector filters. This enables us to wield filters in a very concise and powerful manner. Below I give an example of how you can nest filters to perform complex filtering.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<div>javascript</div>
<div><span class="jQuery">jQuery</span></div>
<div>javascript</div>
<div><span class="jQuery">jQuery</span></div>
<div>javascript</div>
<div><span class="jQuery">jQuery</span></div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

// Select all div's, remove all div's that have a child element with class="jQuery"
alert($('div:not(:has(.jQuery))').text()); // Alerts combined text of all div's

// Select all div's, remove all div's that are odd in the set (count starts at 0)
alert($('div:not(:odd)').text()); // Alerts combined text of all div's

})(jQuery);
</script>
</body>
</html>
```

The take-away concept here is that selector filters can be nested.

> **Notes:**
>
> - You can also nest and stack filters — e.g. `$('p').filter(':not(:first):not(:last)')`

Grokking the `:nth-child()` filter

The `:nth-child()` filter has many uses. For example, say you only want to select every 3rd `<li>` element contained within a `<ul>` element. It is possible with the `:nth-child()` filter. Examine the following code to get a better understanding of how to use the `:nth-child()` filter.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<ul>
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
    <li>5</li>
```

```
    <li>6</li>
    <li>7</li>
    <li>8</li>
    <li>9</li>
    <li>10</li>
</ul>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

// Remember that text() combines the contents of all
// elements in the wrapper set into a single string.

// By index
alert($('li:nth-child(1)').text()); // Alerts "1"

// By even
alert($('li:nth-child(even)').text()); // Alerts "246810"

// By odd
alert($('li:nth-child(odd)').text()); // Alerts "13579"

// By equation
alert($('li:nth-child(3n)').text()); // Alerts "369"

// Remember this filter uses a 1 index
alert($('li:nth-child(0)').text()); // Alerts nothing. There is no 0 index.

})(jQuery);
</script>
</body>
</html>
```

Selecting elements by searching attribute values using regular expressions

When the jQuery attribute filters used to select elements are not robust enough, try using regular expressions. James Padolsey has written a nice extension to the filter selectors that will allow us to create custom regular expressions for filtering. I have provided a code example here, but make sure you also check out the article on http://james.padolsey.com for all the details.

JS Bin: http://jsbin.com/alogo/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<div id="123"></div>
<div id="oneTwoThree"></div>
<div id="0"></div>
<div id="zero"></body>
```

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

//James Padolsey filter extension
jQuery.expr[':'].regex = function(elem, index, match) {
    var matchParams = match[3].split(','),
        validLabels = /^(data|css):/,
        attr = {
            method: matchParams[0].match(validLabels) ?
                        matchParams[0].split(':')[0] : 'attr',
            property: matchParams.shift().replace(validLabels,'')
        },
        regexFlags = 'ig',
        regex = new RegExp(matchParams.join('').replace(/^\s+|\s+$/g,''), regexFlags);
    return regex.test(jQuery(elem)[attr.method](attr.property));
}

// Select div's where the id attribute contains numbers
alert($('div:regex(id,[0-9])').length); // Alerts "2"

// Select div's where the id attribute contains the string "Two"
alert($('div:regex(id, Two)').length); // Alerts "1"

})(jQuery);
</script>
</body>
</html>
```

Difference between selecting direct children vs. all desendants

Direct children elements only can be selected by using the combinator `>` or by way of the
`children()` traversing method. All descendants can be selected by using the `*` css expression.
Make sure you clearly understand the difference between the two. The example below demonstrates
the differences.

JS Bin: http://jsbin.com/izuza/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<div>
<p><strong><span>text</span></strong></p>
<p><strong><span>text</span></strong></p>
</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

// Each statement alerts "2" because there are
```

```
// two direct child <p> elements inside of <div>
alert($('div').children().length);

// or
// alert($('>*', 'div').length);
// alert($('div').find('>*').length);

// Each statement alerts 6 because the <div> contains
// 6 descendants not including the text node.
alert($('div').find('*').length);

// or
// alert($('*', 'div').length);

})(jQuery);
</script>
</body>
</html>
```

Selecting direct child elements when a context is already set

It is possible to use the combinator > without a context to select direct child elements when a context is already being provided. Examine the code below.

```
<!DOCTYPE html>
<html lang="en">
<body>
<ul id="firstUL">
    <li>text</li>
    <li>
        <ul id="secondUL">
            <li>text</li>
            <li>text</li>
        </ul>
    </li>
    <li>text</li>
</ul>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

// Select only the direct <li> children. Alerts "3"
alert($('ul:first').find('> li').length);

// or
// alert($('> li', 'ul:first').length);

})(jQuery);
</script>
```

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

Basically, `'> element'` can be used as an expression when a context has already been determined.

Chapter 3 - Traversing

Difference between `find()` and `filter()` methods

The `filter()` method is used to filter the current set of elements contained within the wrapper set. Its usage should be left to tasks that require filtering a set of elements that are already selected. For example, the code below will filter the three `<p>` elements contained in the wrapper set.

JS Bin: http://jsbin.com/axebo/edit/#html

```html
<!DOCTYPE html>
<html lang="en">
<body>
<p><strong>first</strong></p>
<p>middle</p>
<p><strong>last</strong></p>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

// Alerts middle, by filtering out the first
// and last <p> elements in the wrapper set.
alert($('p').filter(':not(:first):not(:last)').text());

})(jQuery);
</script>
</body>
</html>
```

Notes:

- When using `filter()` always ask yourself if it is absolutely necessary. For example, `$('p').filter(':not(:first):not(:last)')` could be written without `filter()` by passing the jQuery function the expressions as custom selectors `$('p:not(:first):not(:last)')` .

The `find()` method, on the other hand, can be used to further find descendants of the currently selected elements. Think of `find()` more like updating/changing the current wrapped set with new elements that are encapsulated within the elements that are already selected. For example, the code below will change the wrapped set from `<p>` elements to two `<strong>` elements by using `find()`.

JS Bin: http://jsbin.com/ikuno/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<p><strong>first</strong></p>
<p>middle</p>
<p><strong>last</strong></p>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

// Alerts "strong"
alert($('p').find('strong').get(0).nodeName);

})(jQuery);
</script>
</body>
</html>
```

Notes:

- You can actually combine the elements in the wrapper previous to using the `find()` method with the current elements by using `andSelf()` — e.g. `$('p').find('strong').andSelf()`

The take-away concept here is that `filter()` will only reduce (or filter) the currently selected elements in the wrapper set while `find()` can actually create an entirely new set of wrapped elements.

Notes:

- Both `find()` and `filter()` are destructive methods that can be undone by using `end()` - which will revert the wrapped set back to its previous state before `find()` or `filter()` were used.

Passing `filter()` a function instead of an expression

Before you run off and create a custom filter for selecting elements, it might make more sense to simply pass the traversing `filter()` method a function which will allow you to examine each element in the wrapper set for a particular scenario.

For example, let's say that you would like to wrap all `<img>` elements in an html page with an `<p>` element that is currently not wrapped with this element.

You could create a custom filter to accomplish this task or use the `filter()` method by passing it a function that will determine if the element's parent is a `<p>` element and if not then remove it from the set before you wrap the `<img>` elements remaining in the set with a `<p>` element.

I coded below an example where I select every **`<img>`** element in the HTML page then I pass the **`filter()`** method a function that is used to iterate over each element (using **`this`**) in the wrapper set checking to see if the **`<img>`** elements parent element is a **`<p>`** element.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<img />
<img />
<p><img /></p>
<img />
<p><img /></p>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

$('img').attr('src', 'http://static.jquery.com/files/rocker/images/
logo_jquery_215x53.gif')
.filter(function(){return !$(this).parent('p').length == 1}).wrap('<p></p>');

})(jQuery);
</script>
</body>
</html>
```

Notice that I am using **`!`** operater to change a boolean value of true to false. This is because I want to remove **`<img>`** elements from the set that have **`<p>`** elements as their parent element. The function passed to the **`filter()`** method will only remove elements from the set if the function returns false.

The main take-away here is that if you are dealing with an isolated situation, creating a custom filter — e.g. **`:findImgWithNoP`** — for a single situation can be avoided by simply passing the filter method a function that can do custom filtering. This concept is quite powerful. Consider what is possible when we use a regular expressions test in conjunction with the **`filter()`** method.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<ul>
    <li>jQuery is great.</li>
    <li>Its lightweight.</li>
    <li>Its free!</li>
    <li>jQuery makes everything simple</li>
</ul>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){
```

```
// Wrap a <strong> element around any text within
// a <li> that contains the pattern "jQuery"
var pattern = /jQuery/i;
$('ul li').filter(function(){
    return pattern.test($(this).text());
}).wrap('<strong></strong>');

})(jQuery);
</script>
</body>
</html>
```

Traversing up the DOM

You can easily traverse up the DOM to ancestor elements using the `parent()`, `parents()`, and `closest()` methods. Understanding the differences between these methods is critical. Examine the code below and make sure you understand the differences between these jQuery traversing methods.

JS Bin: http://jsbin.com/ukuru/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<div id="parent2">
    <div id="parent1">
        <div id="parent0">
            <div id="start"></div>
        </div>
    </div>
<div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

// Alerts "parent0" x4
alert($('#start').parent().attr('id'));
alert($('#start').parents('#parent0').attr('id'));
alert($('#start').parents()[0].id); // Gets actual DOM element
alert($('#start').closest('#parent0').attr('id'));

// Alerts "parent1" x4
alert($('#start').parent().parent().attr('id'));
alert($('#start').parents('#parent1').attr('id'));
alert($('#start').parents()[1].id); // Gets actual DOM element
alert($('#start').closest('#parent1').attr('id'));

// Alerts "parent2" x4
alert($('#start').parent().parent().parent().attr('id'));
alert($('#start').parents('#parent2').attr('id'));
```

```
alert($('#start').parents()[2].id); // Gets actual DOM element
alert($('#start').closest('#parent2').attr('id'));


})(jQuery);
</script>
</body>
</html>
```

Notes:

- `closest()` and `parents()` might appear to have the same functionality, but `closest()` will actually include the currently selected element in its filtering.

- `closest()` stops traversing once it finds a match, whereas `parents()` gets all parents and then filters on your optional selector. Therefore, `closest()` can only ever return a maximum of 1 element.

Traversing methods accept CSS expressions as optional arguments

CSS expression are not only passed to the jQuery function for selecting elements. They can also be passed to several of the traversing methods. It might be easy to forget this because many of the traversing methods function without having to use any expression at all — e.g. `next()`. The expression is optional for the following traversal methods, but remember that you have the option of providing an expression for filtering.

- childern('expression')
- next('expression')
- nextAll('expression')
- parent('expression')
- parents('expression')
- prev('expression')
- prevAll('expression')
- siblings('expression')
- closest('expression')

Chapter 4 - Manipulation

Creating, operating, and adding HTML on the fly

You can create HTML mark-up on the fly by passing the jQuery function a string of raw HTML.

```
                                                JS Bin: http://jsbin.com/eruca/edit/#html

<!DOCTYPE html>
<html lang="en">
<body>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

alert($('<div><a></a></div>').get(0).nodeName); // Alerts "DIV"
alert($('<div><a></a></div>').length); // Alerts "1" <div> is in the wrapper set
alert($('<div><a></a></div><div><a></a></div>').length); // Alerts "2" two <div> in set

})(jQuery);
</script>
</body>
</html>
```

It is important to note that when creating DOM structures using the jQuery function, only root elements in the structure get added to the wrapper set. In the previous code example the `<div>` elements will be the only elements in the wrapper set.

We can use the `find()` method to operate on any element in the HTML structure once it is created.

```
<!DOCTYPE html>
<html lang="en">
<body>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

$('<div><a></a></div>').find('a').text('jQuery').attr('href', 'http://www.jquery.com');

})(jQuery);
</script>
</body>
</html>
```

After operating on the newly created HTML, it is also possible to add it into the DOM using one of jQuery's manipulation methods. Below we use the `appendTo()` method to add the markup to the page.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

$('<div><a></a></div>')
    .find('a')
    .text('jQuery')
    .attr('href', 'http://www.jquery.com')
    .end().appendTo('body'); // end() is used to exit the find() method

})(jQuery);
</script>
</body>
</html>
```

Notes:

- Simple elements that do not contain attributes — e.g. `$('<div></div>')` — are created via the document.createElement DOM method while all other cases rely on the innerHTML property. In fact, it is possible to directly pass the jQuery function an element created with document.createElement — e.g. `$(document.createElement('div'))`
- The HTML string passed to jQuery cannot contain elements that might be considered invalid inside of a <div> element.
- The HTML string passed to the jQuery function must be well formed.
- You should open and close all HTML elements when passing jQuery HTML. Not doing so could result in bugs, mostly in IE. Just to be safe, always close your HTML elements and avoid writing shortcut HTML — e.g. `$(<div />)`

Grokking the `index()` method

It is possible to determine the index of an element in the wrapper set by passing that element to the `index()` method. As an example, suppose you have a wrapper set containing all the `<div>` elements in a web page and you would like to know the index of the last `<div>` element.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<div>0</div>
```

```
<div>1</div>
<div>2</div>
<div>3</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

// Alerts "3"
alert($('div').index($('div:last')));

// index() will change in 1.3.3 so you can actually do
// $('div:last').index() to return "3" in this example

})(jQuery);
</script>
</body>
</html>
```

The use of `index()` does not really hit home until we consider how it can be used with events. As an example, by clicking the `<div>` elements in the code below we can pass the clicked `<div>` element (using the keyword `this`) to the `index()` method to determine the clicked div's index.

```
<!DOCTYPE html>
<html lang="en">
<body>
<div id="nav">
<div>nav text</div>
<div>nav text</div>
<div>nav text</div>
<div>nav text</div>
<div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

// Alert index of the clicked div amongst all div's in the wrapper set
$('#nav div').click(function(){
    alert($('#nav div').index(this));

// or, nice trick...
    // alert($(this).prevAll().length);
});

})(jQuery);
</script>
</body>
</html>
```

Grokking the `text()` method

One might incorrectly assume that the `text()` method only returns the text node of the first element in a wrapper set. However, it will actually join the text nodes of all elements contained in a wrapper set and then return the concatenated value as a single string. Make sure you are aware of this functionality or you might get some unexpected results.

JS Bin: http://jsbin.com/uxuxa/edit/#html

```html
<!DOCTYPE html>
<html lang="en">
<body>
<div>1,</div>
<div>2,</div>
<div>3,</div>
<div>4</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

alert($('div').text()); // Alerts "1,2,3,4"

})(jQuery);
</script>
</body>
</html>
```

Update or remove characters using a regular expression

Using the JavaScript `replace()` method combined with some jQuery functionality, we can very easily update or remove any pattern of characters from the text contained within an element.

JS Bin: http://jsbin.com/ovava/edit/#html

```html
<!DOCTYPE html>
<html lang="en">
<body>
<p>
    I really hate using JavaScript.
    I mean really hate it!
    It is the best shit ever!
</p>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

var $p = $('p');
// Replace 'hate' with 'love'
```

```
$p.text($p.text().replace(/hate/ig, 'love'));

// Remove 'shit' and replace it with nothing
$p.text($p.text().replace(/shit/ig, ''));

// Keep in mind that text() returns a string, not the jQuery object.
// That is how replace() string method is chained after using text()

})(jQuery);
</script>
</body>
</html>
```

You could also update any characters that are contained in a string returned from `html()`. This means you can not only update text, but also update and replace DOM elements via a regular expression.

Grokking the `.contents()` method

The `.contents()` method can be used to find all the child element nodes, including text nodes contained inside of an element. However, there is a catch. If the retrieved contents contain only text nodes, the selection will placed inside the wrapper set as a single text node. However, if the contents you are retrieving has one or more element nodes amongst the text nodes, then the `.contents()` method will contain text nodes *and* element nodes. Examine the code below to grasp this concept.

```
<!DOCTYPE html>
<html lang="en">
<body>
<p>I love using jQuery!</p>
<p>I love <strong>really</strong> using jQuery!</p>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

// Alerts "I love using jQuery!" because no HTML elements exist
alert($('p:first').contents().get(0).nodeValue);

// Alerts "I love"
alert($('p:last').contents().get(0).nodeValue);

// Alerts "really" but is an HTML element, not text node
alert($('p:last').contents().eq(1).text());

// Alerts "using jQuery!"
alert($('p:last').contents().get(2).nodeValue);

})(jQuery);
```

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

Notice that when an item in the wrapper set is a text node, we have to extract the value by using `.get(0).nodeValue`. The `contents()` method is handy for extracting text node values. It is possible to extract only the text nodes from a DOM structure using `contents()`.

```
<!DOCTYPE html>
<html lang="en">
<body>
<p>jQuery gives me <strong>more <span>power</span></strong> than any other web
tool!</p>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

$('p')
    .find('*') // Select all nodes
    .andSelf() // Include <p>
    .contents() // Grab all child nodes, including text
    .filter(function() {return this.nodeType == Node.TEXT_NODE;}) // Remove non-text
nodes
    .each(function(i,text){alert(text.nodeValue)}); // Alert text contained in wrapper
set

})(jQuery);
</script>
</body>
</html>
```

The `contents()` method also has another use. It gives you access to an HTML document contained within an `<iframe>`, as long as both the `<iframe>` and its parent page reside on the same domain.

```
<!DOCTYPE html>
<html lang="en">
<body>

<iframe src="iframe.html" height="100px" width="100px"></iframe>

<!-- HTML from iframe -->
<!--
<!DOCTYPE html>
<html lang="en">
<body>
<body>
<p>Hi, I am the content inside of the iframe's body element.</p>
</body>
</html>
-->
```

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

// Alerts "<p>Hi, I am the content inside of the iframe's body element.</p>"
alert($('iframe').contents().find('body').html());

// Or, passing the jQuery function a document context...
// Alerts "<p>Hi, I am the content inside of the iframe's body element.</p>"
alert($('body', $('iframe').contents()).html());

})(jQuery);
</script>
</body>
</html>
```

Notes:

- Many have tried to use `contents()` to access `<iframe>` content that is not on the same domain and have been slapped in the face with the same source policy. Make sure you understand the limitations of JavaScript and iframes before you attempt to access or update the content of an iframe. There is a plugin — http://code.google.com/p/jquery-crossframe/ — which contains a work-around if you want to attempt cross-domain `<iframe>` communication.

Using `remove()` does not remove elements from wrapper set

When you `remove()` a DOM snippet from the DOM the elements contained in the removed DOM structure are still contained within the wrapper set. You could remove an element, operate on that element, and then actually place that element back into the DOM, all within a single jQuery chain.

JS Bin: http://jsbin.com/icayi/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<div>remove me</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

$('div').remove().html('<a href="http://www.jQuery.com">jQuery</a>').appendTo('body');

})(jQuery);
</script>
</body>
</html>
```

The take-away here is that just because you `remove()` elements does not mean they are removed from the jQuery wrapper set.

Chapter 5 - HTML Forms

Disable/enable form elements

Using jQuery we can easily disable form elements by setting the disabled attribute value of a form element to disabled. To do this we simply select an input, then using the **attr()** method we set the disabled attribute of the input to a value of disabled.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<input name="button" type="button" id="button" />
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

$('#button').attr('disabled', 'disabled');

})(jQuery);
</script>
</body>
</html>
```

To enable a disabled form element, we simply remove the disabled attribute using **removeAttr()** or set the disabled attribute value to be empty using **attr()**.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<input name="button" type="button" id="button" value='button' disabled="disabled" />
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

$('#button').removeAttr('disabled');

// or
// $('#button').attr('disabled', '');

})(jQuery);
</script>
```

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

How to determine if a form element is disabled or enabled

Using the jQuery form filter expressions `:disabled` or `:enabled` it is rather easy to select and determine (boolean value) if a form element is disabled or enabled. Examine the code below for clarity.

JS Bin: http://jsbin.com/ehiva/edit/#html

```html
<!DOCTYPE html>
<html lang="en">
<body>
<input name="button" type="button" id="button1" />
<input name="button" type="button" id="button2" disabled="disabled" />
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

// Is it enabled?
alert($('#button1').is(':enabled')); // Alerts true

// Or, using a filter
alert($('#button1:enabled').length); // Alerts "1"

// Is it disabled?
alert($('#button2').is(':disabled')); // Alerts "true"

// Or, using a filter
alert($('#button2:disabled').length); // Alerts "1"

})(jQuery);
</script>
</body>
</html>
```

Check/uncheck a single checkbox or radio button

You can check a radio input or checkbox by setting its `checked` attribute to `true` using the `attr()`.

JS Bin: http://jsbin.com/iditu/edit/#html

```html
<!DOCTYPE html>
<html lang="en">
```

```
<body>
<input name="" value="" type="checkbox" />
<input name="" value="" type="radio" />
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

// Set all checkboxes or radio buttons to checked
$('input:checkbox,input:radio').attr('checked', 'checked');


})(jQuery);
</script>
</body>
</html>
```

To uncheck a radio input or checkbox, simply remove the checked attribute using the `removeAttr()` method or set the `checked` attribute value to an empty string.

```
<!DOCTYPE html>
<html lang="en">
<body>
<input name="" type="checkbox" value="" checked="checked">
<input name="" type="radio" value="" checked="checked">
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

$('input').removeAttr('checked');

// or
$('input').attr('checked', '');

})(jQuery);
</script>
</body>
</html>
```

Check/uncheck multiple checkboxs or radio inputs

You can use the jQuery `val()` on multiple checkbox inputs or radio inputs to set these inputs to checked. This is done by passing the `val()` method an array containing a string which coincides with the checkbox input or radio input value attribute.

```
<!DOCTYPE html>
<html lang="en">
```

```
<body>
<input type="radio" value="radio1">
<input type="radio" value="radio2">
<input type="checkbox" value="checkbox1">
<input type="checkbox" value="checkbox2">
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

// Check all radio and checkbox inputs on the page.
$('input:radio,input:checkbox').val(['radio1', 'radio2', 'checkbox1', 'checkbox2']);

// Use explict iteration to uncheck.
// $('input:radio,input:checkbox').removeAttr('checked');

// or
// $('input:radio,input:checkbox').attr('checked', '');

})(jQuery);
</script>
</body>
</html>
```

Notes:

- If the checkbox or radio button is already checked, using `val()` will not uncheck the input element.

Determining if a checkbox or radio button is checked or unchecked

We can determine if a checkbox input or radio input is checked or unchecked by using the `:checked` form filter. Examine the code below for several usages of the `:checked` filter.

JS Bin: http://jsbin.com/izote/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<input checked="checked" type="checkbox" />
<input checked="checked" type="radio" />
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

// Alerts "true"
alert($('input:checkbox').is(':checked'));

// Or, added to wrapper set if checked. Alerts "1"
alert($('input:checkbox:checked').length);
```

```
// Alerts "true"
alert($('input:radio').is(':checked'));

// Or, added to wrapper set if checked. Alerts "1"
alert($('input:radio:checked').length);

})(jQuery);
</script>
</body>
</html>
```

How to determine if a form element is hidden

We can determine if a form element is hidden using the `:hidden` form filter. Examine the code below for several usages of the `:hidden` filter.

JS Bin: http://jsbin.com/aniso/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<input type="hidden" />
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

// Alerts "true"
alert($('input').is(':hidden'));

// Or, added to wrapper set if hidden. Alerts "1"
alert($('input:hidden').length);

})(jQuery);
</script>
</body>
</html>
```

Setting/getting the value of an input element

The `val()` method can be used to set and get the attribute value of an input element (button, checkbox, hidden, image, password, radio, reset, submit, text). Below I set the value for each input in `val()` and then alert the value using the `val()` method.

JS Bin: http://jsbin.com/ibura/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<input type="button" />
<input type="checkbox" />
<input type="hidden" />
<input type="image" />
<input type="password" />
<input type="radio" />
<input type="reset" />
<input type="submit" />
<input type="text" />
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

$('input:button').val('I am a button')
$('input:checkbox').val('I am a checkbox')
$('input:hidden').val('I am a hidden input')
$('input:image').val('I am a image')
$('input:password').val('I am a password')
$('input:radio').val('I am a radio')
$('input:reset').val('I am a reset')
$('input:submit').val('I am a submit')
$('input:text').val('I am a text')

// Alerts input's value attribute
alert($('input:button').val());
alert($('input:checkbox').val());
alert($('input:hidden').val());
alert($('input:image').val());
alert($('input:password').val());
alert($('input:radio').val());
alert($('input:reset').val());
alert($('input:submit').val());
alert($('input:text').val());

})(jQuery);
</script>
</body>
</html>
```

Setting/getting the selected option of a select element

Using the `val()` method we can set the selected value of a `<select>` element by passing the `val()` method a string representing the text node contained within the `<option>` element.

To get the value of the `<select>` element we use again the `val()` method to determine which option is selected. The `val()` method in this scenario will return the selected option's attribute value.

```
<!DOCTYPE html>
<html lang="en">
<body>
<select>
    <option value="option1">option one</option>
    <option value="option2">option two</option>
</select>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

// Set the selected option in the select element to "option two"
$('select').val('option two');

// Alerts "option2"
alert($('select').val());

})(jQuery);
</script>
</body>
</html>
```

Setting/getting the selected options of a multi-select element

Using the `val()` method we can set the selected values of a multi-select element by passing the `val()` method an array containing the corresponding text nodes.

To get the selected options in a multi-select element, we again use the `val()` method to retrieve an array of the options that are selected. The array will contain the value attributes of the selected options.

```
<!DOCTYPE html>
<html lang="en">
<body>

<select size="4" multiple="multiple">
    <option value="option1">option one</option>
    <option value="option2">option two</option>
    <option value="option3">option three</option>
    <option value="option4">option four</option>
</select>

<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){
```

```
// Set the value of the selected options
$('select').val(['option two', 'option four']);

// Get the selected values
alert($('select').val().join(', ')); // Alerts, "option2, option4"

})(jQuery);
</script>
</body>
</html>
```

Setting/getting text contained within a textarea

You can set the text node contents of a `<textarea>` element by passing the `val()` method a text string to be used as the text. To get the value of a textarea element we again use the `val()` method to retrieve the text contained within.

JS Bin: http://jsbin.com/eyiso/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<textarea></textarea>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

// Set the text contained within
$('textarea').val('I am a textarea');

// Alerts "I am a textarea"
alert($('textarea').val());

})(jQuery);
</script>
</body>
</html>
```

Setting/getting the value attribute of a button element

You can set the value attribute of a button element by passing the `val()` method a text string. To get the value of a button element we again use the `val()` method to retrieve the text.

JS Bin: http://jsbin.com/afoye/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<button>Button</button>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

// Set the value: <button value="I am a Button Element">
$('button').val('I am a Button Element')

// Alerts "I am a Button Element"
alert($('button').val());

})(jQuery);
</script>
</body>
</html>
```

Editing select elements

jQuery makes some of the common tasks associated with editing select elements trivial. Below are some of those tasks with coded examples.

```
// Add options to a select element at the end
$('select').append('<option value="">option</option>');
```

```
// Add options to the start of a select element
$('select').prepend('<option value="">option</option>');
```

```
// Replace all the options with new options
$('select').html('<option value="">option</option><option value="">option</option>');
```

```
// Replace items at a certain index using :eq() selecting filter to
// select the element then replace it with .replaceWith() method
$('select option:eq(1)').replaceWith('<option value="">option</option>');
```

```
// Set the select elements' selected option to index 2
$('select option:eq(2)').attr('selected', 'selected');
```

```
// Remove the last option from a select element
$('select option:last').remove();
```

```
// Select an option from a select element via its
// order in the wrapper set using custom filters
$('#select option:first');
$('#select option:last');
$('#select option:eq(3)');
$('#select option:gt(5)');
$('#select option:lt(3)');
$('#select option:not(':selected')');
```

```
// Get the text of the selected option(s), this will return the text of
// all options that are selected when dealing with a muli-select element
$('select option:selected').text();
```

```
// Get the value attribute value of an option in a select element
$('select option:last).val(); // Getting the :last option element
```

```
// Get the index (0 index) of the selected option.
// Note: Does not work with multi-select elements.
$('select option').index($('select option:selected'));
```

```
// Insert an option after a particular position
$('select option:eq(1)').after('<option value="">option</option>');
```

```
// Insert an option before a particular position
$('select option:eq(3)').before('<option value="">option</option>');
```

Selecting form elements by their type

It is possible to select form elements by their type — e.g. `$('input:checkbox')`. jQuery provides the following form type filters for selecting form element by their type.

- :text
- :password
- :radio
- :checkbox
- :submit
- :image

- :reset
- :file
- :button

Selecting all form elements

It is possible to select all form elements by using the `:input` form filter. This filter will select more than just input elements, it will select any `<textarea>`, `<select>`, or `<button>` elements as well. In the coded example below take notice of the length of the wrapper set when using the `:input` filter.

JS Bin: http://jsbin.com/ikeki/edit/#html

```html
<!DOCTYPE html>
<html lang="en">
<body>
<input type="button" value="Input Button"/>
<input type="checkbox" />
<input type="file" />
<input type="hidden" />
<input type="image" />
<input type="password" />
<input type="radio" />
<input type="reset" />
<input type="submit" />
<input type="text" />
<select><option>Option</option></select>
<textarea></textarea>
<button>Button</button>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

// Alerts "13" form elements
alert($(':input').length);

})(jQuery);
</script>
</body>
</html>
```

Chapter 6 - Events

Not limited to a single `ready()` event

It is important to keep in mind that it is possible to declare as many custom `ready()` events as you would like. You are not limited to attaching a single `.ready()` event to the document. The `ready()` events are executed in the order that they are included.

> **Notes:**
>
> - Passing the jQuery function, a function — e.g. `jQuery(funciton(){//code here})` — is a shortcut for `jQuery(document).ready()`.

Attaching/removing events using `bind()` and `unbind()`

Using the `bind()` method — e.g `jQuery('a').bind('click',function(){})` — we can add any of the follow standard handlers to the appropriate DOM elements.

- blur
- focus
- load
- resize
- scroll
- unload
- beforeunload
- click
- dblclick
- mousedown
- mouseup
- mousemove
- mouseover
- mouseout
- change
- select
- submit
- keydown

- [keypress](#)
- [keyup](#)
- [error](#)

Obviously, based on DOM standards, only certain handlers coincide with particular elements.

In addition to this list of standard handlers, you can also leverage **bind()** to attach jQuery custom handlers — e.g. **mouseenter** and **mouseleave** — as well as any custom handlers you may create.

To remove standard handlers or custom handlers, we simply pass the **unbind()** method the handler name or custom handler name that needs to be removed — e.g. **jQuery('a').unbind('click')**. If no parameters are passed to **unbind()** it will remove all handlers attached to an element.

These concepts just discussed are expressed in the code example below.

JS Bin: http://jsbin.com/ulezi/edit/#html

```html
<!DOCTYPE html>
<html lang="en">
<body>

<input type="text" value="click me" />
<br />
<br />
<button>remove events</button>

<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

// Bind events
$('input').bind('click', function(){
    alert('You clicked me!');
});

$('input').bind('focus', function(){
    alert('You focused this input!');
});

// Unbind events
$('button').click(function(){ // Using shortcut binding via click()
    $('input').unbind('click');
    $('input').unbind('focus');

  // Or, unbind all events
    // $('a:first').unbind();
});

})(jQuery);
</script>
</body>
</html>
```

Programmatically invoke a specific handler via short event methods

The shortcut syntax — e.g. `.click(), mouseout(), focus()` — for binding an event handler to a DOM element can also be used to invoke handlers programmatically. To do this simply use the shortcut event method without passing it a function. In theory, this means that we can bind a handler to a DOM element and then immediately invoke that handler. Below I do demonstrate via the `click()` event.

JS Bin: http://jsbin.com/ugujo/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<a>Say Hi</a> <!-- click this element will alert "hi" -->
<a>Say Hi</a> <!-- click this element will alert "hi" -->
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

// Bind a click handler to all <a> and immediately invoke their handlers
$('a').click(function(){ alert('hi') }).click();

// Page will alert twice. On page load, a click
// is triggered for each <a> in the wrapper set.

})(jQuery);
</script>
</body>
</html>
```

Notes:

- It is also possible to use the event `trigger()` method to invoke specific handlers — e.g. `jQuery('a').click(function(){ alert('hi') }).trigger('click')`. This will also work with namespaced and custom events.

jQuery normalizes the event object

[jQuery normalized the event object](#) according to W3C standards. This means that when the event object is passed to a function handler you do not have to worry about [browser specific implementations of the event object](#) (e.g. IE's `window.event`) . You can use the following attributes and methods of the event object worry-free from browser differences because jQuery normalizes the event object.

Event object Attributes
- event.type
- event.target
- event.data
- event.relatedTarget
- event.currentTarget
- event.pageX
- event.pageY
- event.result
- event.timeStamp

Event object Methods
- event.preventDefault()
- event.isDefaultPrevented()
- event.stopPropagation()
- event.isPropagationStopped()
- event.stopImmediatePropagation()
- event.isImmediatePropagationStopped()

To acces the normalized jQuery event object, simply pass the anonymous function, passed to a jQuery event method, a parameter named "event" (or whatever you want to call it). Then, inside of the anonymous callback function use the parameter to access the event object. Below is a coded example of this concept.

JS Bin: http://jsbin.com/unucu/edit/#html

```html
<!DOCTYPE html>
<html lang="en">
<body>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

$(window).load(function(event){alert(event.type);}); // Alerts "load"

})(jQuery);
</script>
</body>
</html>
```

Grokking event namespacing

Often we will have an object in the DOM that needs to have several functions tied to a single event handler. For example, let's take the resize handler. Using jQuery we can add as many functions to the `window.resize` handler as we like. But what happens when we need to remove only one of these functions but not all of them? If we use `$(window).unbind('resize')` it will remove all functions attached to the `window.resize` handler. By namespaceing a handler (e.g. `resize.unique`) we can assign a unique hook to a specific function for removal.

JS Bin: http://jsbin.com/ikosa/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

$(window).bind('resize', function(){ alert('I have no namespace');});
$(window).bind('resize.unique', function(){ alert('I have a unique namespace');});

// Removes only the resize.unique function from event handler
$(window).unbind('resize.unique')

})(jQuery);
</script>
</body>
</html>
```

In the above code we are adding two functions to the resize handler. The second (document order) resize event added uses event namespacing and then immediately removes this event using `unbind()`. I did this to make the point that the first function attached is not removed. Namespacing events gives us the ability to label and remove unique functions assigned to the same handler on a single DOM element.

In addition to unbinding a specific function associated with a single DOM element and handler, we can also use event namespacing to exclusively invoke (using `trigger()`) a specific handler and function attached to a DOM element. In the code below two click events are being added to the `<a>` and then using namespacing only one is invoked.

JS Bin: http://jsbin.com/atixu/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<a>click</a>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){
```

```
$('a').bind('click',function(){alert('You clicked me')});
$('a').bind('click.unique',function(){alert('You Trigger click.unique')});

// Invoke the function passed to click.unique
$('a').trigger('click.unique');

})(jQuery);
</script>
</body>
</html>
```

Notes:

- There is no limit to the depth or number of namespaces used — e.g. `resize.layout.headerFooterContent`
- Namespacing is a great way of protecting, invoking, removing any exclusive handlers that a plugin may require
- Namespacing works with custom events as well as standard events — e.g. `click.unique` or `myclick.unique`

Grokking event delegation

Event delegation relies on event propagation (a.k.a bubbling). When you click an `<a>` in side of a `<li>`, which is inside of a `<ul>`, the click event bubbles up the DOM from the `<a>` to the `<li>` to the `<ul>` and so on, until each ancestor element with a function assigned to an event handler fires.

This means if we attach a click event to a `<ul>` and then click an `<a>` that is encapsulated inside of the `<ul>` eventually the click handler attached to the `<ul>`, because of bubbling, will get invoked. When it does get invoked we can use the event object (`event.target`) to identify which element in the DOM actually caused the event bubbling to begin. Again, this will give us a reference to the element that started the bubbling.

By doing this we can seemly add an event handler to a great deal of DOM elements using only a single event handler/declaration. This is extremely useful, for example, a table with 500 rows where each row requires a click event can take advantage of event delegation. Examine the code below for clarifications.

JS Bin: http://jsbin.com/alane/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<ul>
    <li><a href="#">remove</a></li>
    <li><a href="#">remove</a></li>
    <li><a href="#">remove</a></li>
    <li><a href="#">remove</a></li>
    <li><a href="#">remove</a></li>
```

```
    <li><a href="#">remove</a></li>
</ul>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

$('ul').click(function(event){ // Attach click handler to <ul> and pass event object
    // event.target is the <a>
    $(event.target).parent().remove(); // Remove <li> using parent()
    return false; // Cancel default browser behavior, stop propagation
});

})(jQuery);
</script>
</body>
</html>
```

Now, if you were to literally click on one of the actual bullets of the list and not the link itself, guess what? You'll end up removing the `<ul>`. Why? Because all clicks bubble. So when you click on the bullet, the `event.target` is the `<li>` not the the `<a>`. Since this is the case the `parent()` method will grab the `<ul>` and remove it. We could update our code so that we only remove an `<li>` when it is being clicked from an `<a>` by passing the the `parent()` method an element expression.

```
$(event.target).parent('li').remove();
```

The important take-away here is that you have to manage carefully what is being clicked on when the click-able area contains multiple encapsulated elements due to the fact that you never know exactly where the user may click. Because of this you have to check to make sure the click occurred from the element you expected it to.

Applying event handlers to DOM elements regardless of DOM updates using `live()`

Using the handy `live()` event method we can bind handlers to DOM elements currently in a web page and those that have yet to be added. The `live()` method uses event delegation to make sure that newly added/created DOM elements will always respond to event handlers regardless of DOM manipulations or dynamic changes to the DOM. Using `live()` is essentially a shortcut for manually having to set up event delegation. For example, using `live()` we could create a button that creates another button indefinitely.

JS Bin: http://jsbin.com/ikaqo/edit/#html

```
<!DOCTYPE html>
<html lang="en">
```

```
<body>
<button>Add another Button</button>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

$('button').live('click',function(){
    $(this).after("<button>Add another Button</button>");
});

})(jQuery);
</script>
</body>
</html>
```

After examining the code it should be obvious that we are using `live()` to apply event delegation to a parent element (`<body>` element in the code example) so that any button element added to the DOM always responds to the click handler.

To remove the live event we simply use the `die()` method — e.g. `$('button').die()`.

The take-away concept here is the `live()` method could be used to attach events to DOM elements that are removed and added using AJAX. In this way you would forgo having to rebind events to new elements introduced into the DOM after the initial page load.

> **Notes:**
>
> - `live()` supports the following handlers: `click`, `dblclick`, `mousedown`, `mouseup`, `mousemove`, `mouseover`, `mouseout`, `keydown`, `keypress`, `keyup`.
> - `live()` only work agianst a selector.
> - `live()` by default will stop propagation by using `return false` within the function sent to the `live()` method.

Adding a function to several event handlers

It is possible to pass the event `bind()` method several event handlers. This makes it possible to attach the same function, written once, to many handlers. In the code example below we attach a single anonymous callback function to the click, keypress, and resize event handlers on the document.

JS Bin: http://jsbin.com/azisi/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
```

```
(function($){

// Responds to multiple events
$(document).bind('click keypress resize', function(event){
    alert('A click, keypress, or resize event occured on the document.');
});


})(jQuery);
</script>
</body>
</html>
```

Cancel default browser behavior with `preventDefault()`

When a link is clicked or a form is submitted the browser will invoke its default functionality associated with these events. For example, click an `<a>` link and the web browser will attempt to load the value of the `<a> href` attribute in the current browser window. To stop the browser from performing these types of functionality we can use the `preventDefault()` method of the jQuery normalized event object.

```
                                                    JS Bin: http://jsbin.com/acowo/edit/#html
<!DOCTYPE html>
<html lang="en">
<body>
<a href="http://www.jquery.com">jQuery</a>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

// Stops browser from navigating
$('a').click(function(event){
    event.preventDefault();
});


})(jQuery);
</script>
</body>
</html>
```

Cancel event propagation with `stopPropagation()`

Events propagate (a.k.a bubbling) up the DOM. When an event handler is fired for any given element the invoked event handler is also invoked for all ancestor elements. This default behavior facilitates

solutions like event delegation. To prohibit this default bubbling we can use the jQuery normalized event method `stopPropagation()` to negate this behavior.

```html
<!DOCTYPE html>
<html lang="en">
<body>

<p><span>stop</span></p>

<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

    $('p').click(function(){window.location = 'http://www.jquery.com'});

    $('span').click(function(event){
    alert('You clicked a span inside of an p element');
    event.stopPropagation(); //remove this and the click event will bubble to the <p>
    });

})(jQuery);
</script>
</body>
</html>
```

In the coded example above the `<a>` will not link to jquery.com because we stop propagation from bubbling up from the `<span>` element to the `<a>` element. Go ahead click the stop link, nothing happens.

Cancel default browser behavior and event propagation via `return false`

Returning false — e.g. `return false` — is the equivalent of using both `preventDefault()` and `stopPropagation()`.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<span><a href="javascript:alert('You clicked me!')" class="link">click me</a></span>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){
    $('span').click(function(){ // Add click event to <span>
```

```
        window.location='http://www.jquery.com';
    });

    $('a').click(function(){ // Ignore clicks on <a>
    return false;
    });
})(jQuery);
</script>
</body>
</html>
```

If you were to comment out the **return false** statement in the code above the the **alert()** would get invoked because by default the browser will execute the value of the **href**. Also, the page would navigate to jquery.com due to event bubbling.

Create custom events and trigger them via `trigger()`

With jQuery you have the ability to manufacture your own custom events using the **bind()** method. This is done by providing the **bind()** method with a unique name for a custom event.

Now, because these events are custom and not known to the browser, the only way to invoke custom events is to programmatically trigger them using the jQuery **trigger()** method. Examine the code below for an example of a custom event that is then invoked using **trigger()**.

JS Bin: http://jsbin.com/icone/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<div>jQuery</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

$('div').bind('myCustomEvent', function(){ // Bind a custom event to <div>
    window.location = 'http://www.jquery.com';
});

$('div').click(function(){ // Click the <div> to invoke the custom event
    $(this).trigger('myCustomEvent');
})

})(jQuery);
</script>
</body>
</html>
```

Cloning events as well as DOM elements

By default cloning DOM structures using the `clone()` method does not additionaly clone the events attached to the DOM elements being cloned. In order to clone the elements and the events attached to the elements you must pass the `clone()` method a boolean value of `true`.

```html
                                                JS Bin: Blows up on this one!

<!DOCTYPE html>
<html lang="en">
<body>
<button>Add another Button</button>
<a href="#" class="clone">Add another Link</a>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

$('button').click(function(){
    var $this = $(this);
    $this.clone(true).insertAfter(this); // Clone element and its events
    $this.text('button').unbind('click'); // Change text, remove event
});

$('.clone').click(function(){
    var $this = $(this);
    $this.clone().insertAfter(this); // Clone element, but not its events
    $this.text('link').unbind('click'); // Change text, remove event
});

})(jQuery);
</script>
</body>
</html>
```

Using Firebug to reveal/inspect events attached to DOM elements

A handy trick using the firebug console is the ability to print out an interactive listing of the events that have been attached to a DOM element using jQuery. In the code below a `click` and `mouseleave` event is being attached to the only `<div>` element in the page. Using Firebug we can display and inspect the events attached to that `<div>` element by accessing the event data stored (`$('div').data('events')`) for the `<div>` element.

```html
                                                JS Bin: http://jsbin.com/apori/edit/#html

<!DOCTYPE html>
<html lang="en">
```

```
<body>
<div>click or hover over this text</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

$('div').bind('click mouseleave', function(event){
    alert(event.type);
});

// Open Firebug and look at the console to see the attached events
console.dir($('div').data('events'));

})(jQuery);
</script>
</body>
</html>
```

Event information is stored on elements when the events are bound.

Getting X and Y coordinates of the mouse in the viewport

By attaching a `mousemove` event to the entire page (document) we can retrive the X and Y coordinates of the mouse pointer as it moves around inside in the viewport over the canvas. This is done by retrieving the pageY and pageX properties of the jQuery normalized event object.

```
                                        JS Bin: http://jsbin.com/ohawa/edit/#html
<!DOCTYPE html>
<html lang="en">
<body>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

$(document).mousemove(function(e){
    // e.pageX - gives you X position
    // e.pageY - gives you Y position
    $('body').html('e.pageX = ' + e.pageX + ', e.pageY = ' + e.pageY);
});

})(jQuery);
</script>
</body>
<html>
```

Getting X and Y coordinates of the mouse relative to another element

It is often neccesary to get the X and Y coordinates of the mouse pointer relative to an element other than the viewport/entire document. This is often done with tooltips, where the tooltip is shown relative to the location that the mouse is hovering. This can easly be accomplished by subtracting the offset of the relative element from the viewports X and Y mouse coordinates.

JS Bin: http://jsbin.com/olanu/edit/#html

```html
<!DOCTYPE html>
<html lang="en">
<body>
<!-- Move mouse over div to get position relative to the div -->
<div style="margin:200px;heigth:100px;width:100px;background:#ccc;padding:20px">
    relative to this
</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

$('div').mousemove(function(e){
    //relative to this div element instead of document
    var relativeX = e.pageX - this.offsetLeft;
    var relativeY = e.pageY - this.offsetTop;
    $(this).html('releativeX = ' + relativeX + ', releativeY = ' + relativeY);
});

})(jQuery);
</script>
</body>
</html>
```

Chapter 7 - jQuery and the web browser

Disable right-click contextual menu

Using JavaScript, it is possible to disable the browsers right-click native [contextual menu](). Doing so with jQuery is a snap. We simply cancel the `contenxtmenu` event.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

$(document).bind('contextmenu', function(){
    return false;
});

})(jQuery);
</script>
</body>
</html>
```

Notes:

- Does not work in Opera.

Scrolling the browser window

While there are numerous plugins for scrolling the browser window, doing so can be trivial when a simple scroll is required. By setting the `scrollTop` CSS property on the `<html>` and `<body>` element it is possible to control the position of the horizontal (or, vertical) scroll. In the code below I use the `animate()` method to animate the horizontal scrolling to a specific element in the page.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<style>
li {
    padding-bottom: 500px;
}
</style>
<ul>
<li><a href="#" class="next">Next</a></li>
<li><a href="#" class="next">Next</a>/<a href="#" class="prev">Previous</a></li>
<li><a href="#" class="next">Next</a>/<a href="#" class="prev">Previous</a></li>
<li><a href="#" class="prev">Previous</a></li>
</ul>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

$('.next').click(function(){
    $('html, body').animate({
scrollTop: $(this).parent().next().find('a').offset().top},
        1000);
});

$('.prev').click(function(){
    $('html, body').animate({
        scrollTop: $(this).parent().prev().find('a').offset().top},
        1000);
});

})(jQuery);
</script>
</body>
</html>
```

Chapter 8 - Plugins

Use the $ alias when constructing a plugin

When writing a jQuery plugin, the same conflict prevention routine used with regular-old jQuery code should be implemented. With this in mind, all plugins should be contained inside a private scope where the $ alias can be used without fear of conflicts or surprising results.

The coding structure below should look familiar as it is used in almost every code example in this book and explained in Chapter 1.

```html
JS Bin: http://jsbin.com/axuze/edit/#html

<!DOCTYPE html>
<html lang="en">
<body>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
alert(jQuery(document).jquery); // Don't use $ here. It is not reliable.

(function($){

    // Can use $ without fear of conflicts
    alert($(document).jquery);

})(jQuery);

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

New plugins attach to `jQuery.fn` object to become jQuery methods

New plugins are attached to the `jQuery.fn` object, as this is a shortcut or alias for `jQuery.prototype`. In our coding example below we are adding the count plugin to the `jQuery.fn` object. By doing this we are creating our own custom jQuery method that can be used on a wrapped set of DOM elements .

Basically, a plugin attached to `jQuery.fn` allows us to create our own custom methods similar to any found in the API. This is because when we attach our plugin function to `jQuery.fn` our function

is included in the prototype chain — `$.fn.count = function(){}` — for jQuery objects created using the jQuery function. If that blows your mind, just remember that adding a function to `jQuery.fn` means that the keyword `this` inside of the plugin function will refer to the jQuery object itself.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<div id="counter1"></div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

$.fn.count = function(){
    var $this = this; // "this" is the jQuery object
    $this.text('0'); // Sets the counter start number to zero
    var myInterval = window.setInterval(function(){ // Interval for counting
        var currentCount = parseFloat($this.text());
        var newCount = currentCount+1;
        $this.text(newCount+'');
    }, 1000);
};

})(jQuery);

jQuery('#counter1').count();

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

Notes:

- By adding a plugin to the `jQuery.fn` object we are essentially saying that our plugin would like to use the jQuery function to select a context (DOM elements). If your plugin does not require a specific context (in other words a set of DOM elements) in which it needs to operate you might not need to attach this plugin to the `$.fn`. It might make more sense to add it as a utility function in the jQuery namespace.

Inside a plugin, `this` is a reference to the current jQuery object

When you attach a plugin to the `jQuery.fn` object the keyword `this` used inside of the attached plugin function will refer to the current jQuery object.

```
<!DOCTYPE html>
<html lang="en">
<body>
<div id="counter1"></div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($) {

$.fn.count = function(){
    // "this" is equal to jQuery('#counter1')
    alert(this); // Alerts jQuery object
    alert(this[0]); // Alerts div element
    alert(this[0].id); // Alerts "counter1"
};

})(jQuery);

jQuery('#counter1').count();

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

It is critical that you grok exactly what the keyword `this` is referring to in the plugin function.

`each()` is used to iterate over the jQuery object and provide a reference to each element in the object using the `this` keyword

Using `each()` we can create implicit iteration for our plugin. This means that if the wrapper set contains more than one element our plugin method will be applied to each element in the wrapper set.

To do this we use the jQuery utility `each()` function, which is a generic iterator for both objects and arrays. It basically simplifies looping. In the code example below we use it to iterate over the jQuery object itself. Inside of the function that is passed to `each()` the keyword `this` will refer to the elements in the jQuery wrapper set.

```
<!DOCTYPE html>
<html lang="en">
<body>
<div id="counter1"></div>
<div id="counter2"></div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>

(function($) {
```

```
    $.fn.count = function(){
        this.each(function(){ // "this" is the current jQuery object
            var $this = $(this);
            $this.text('0'); // Sets the counter start number
            var myInterval = window.setInterval(function(){ // Interval for counting
                var currentCount = parseFloat($this.text());
                var newCount = currentCount+1;
                $this.text(newCount + '');
            }, 1000);
        });
    };
})(jQuery);

jQuery('#counter1, #counter2').count();

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

Using the `each()` function is critical if you would like a plugin to employee implicit iteration.

Typically a plugin returns the jQuery object so jQuery methods or other plugins can still be chained after using a plugin

It is typical that most plugins will return the jQuery object itself so that the plugin does not break the chain. In other words, if a plugin does not specifically need to return a value it should continue the chain so that additional methods can be applied to the wrapper set. In the code below we are returning the jQuery object with the, `return this;` statement so that that chaining will not be broken. Notice that I am chaining on the `parent()` and `append()` method after I call the `count()` plugin.

```
                                        JS Bin: http://jsbin.com/ebiro/edit/#html

<!DOCTYPE html>
<html lang="en">
<body>
<div id="counter1"></div>
<div id="counter2"></div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>

(function($) {
    $.fn.count = function(){
        return this.each(function(){ // Return the jQuery object, or "this" after
each()
        var $this = $(this);
            $this.text('0');
            var myInterval = window.setInterval(function(){
```

91

```
                    var currentCount = parseFloat($this.text());
                    var newCount = currentCount+1;
                    $this.text(newCount + '');
                }, 1000);
            });
        };
    })(jQuery);


    jQuery('#counter1, #counter2')
        .count()
    .parent() // Chaining continues because jQuery object is returned
        .append('<p>Chaining still works!</p>');


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

Default plugin options

Plugins typically contain default options that will act as the baseline default configuration for the plugins logic. These options are used when the plugin is invoked. In the code below I am creating a **defaultOptions** object containing a single property (**startCount)** and value (**0**). This object is stored on the count function, **$.fn.count.defaultOptions**. We do this so that the options are configurable from outside of the plugin.

JS Bin: http://jsbin.com/atevu/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<div id="counter1"></div>
<div id="counter2"></div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>

(function($) {
    $.fn.count = function(){
        return this.each(function(){
            var $this = $(this);
            // Sets the counter start number to zero
            $this.text($.fn.count.defaultOptions.startCount + '');
            var myInterval = window.setInterval(function(){
                var currentCount = parseFloat($this.text());
                var newCount = currentCount + 1;
```

```
                $this.text(newCount + '');
            }, 1000);

        });
    };

$.fn.count.defaultOptions = {
        startCount: 100
    };
})(jQuery);

jQuery('#counter1, #counter2').count();

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

Custom plugin options

Typically, the default plugin options can be overwritten with custom options. In the code below we are passing in a `customOptions` object as a parameter to the plugin function. This object is combined with the `defaultOptions` object to create a single `options` object. We are using the jQuery utility method `extend()` to combine multiple objects into a single object. The `extend()` method provides the perfect utility for overwriting an object with new properties. With this code in place, the plugin can now be customized when invoked. In the example we are passing the `count` plugin a custom number (500) to be used as the starting point for the count. This custom option overrides the default option (0).

```
<!DOCTYPE html>
<html lang="en">
<body>
<div id="counter1"></div>
<div id="counter2"></div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>

(function($) {
    $.fn.count = function(customOptions){
        // Create new option, extend object with defaultOptoins and customOptions
        var options = $.extend({},$.fn.count.defaultOptions, customOptions);
        return this.each(function(){
            var $this = $(this);
        // Sets the counter start number to the default option value
            // or to custom option value if it is passed to the plugin
            $this.text(options.startCount + '');
            var myInterval = window.setInterval(function(){
                var currentCount = parseFloat($this.text());
```

```
            var newCount = currentCount+1;
            $this.text(newCount+'');
        }, 1000);
    });
};

$.fn.count.defaultOptions = {
    startCount:100
};
})(jQuery);


// Passing a custom option overrides default
jQuery('#counter1, #counter2').count({startCount:500});


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

Overwrite default options without altering original plugin code

Since default options are accessible from outside of a plugin, it is possible to reset the default options before invoking the plugin. This can be handy when you want to define your own options without altering the plugin code itself. Doing so can simplify plugin invocations because you can, in a sense, globally setup the plugin to your liking without forking the original plugin code itself.

```
<!DOCTYPE html>
<html lang="en">
<body>
<div id="counter1"></div>
<div id="counter2"></div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>

(function($) {
    $.fn.count = function(customOptions){
        var options = $.extend({},$.fn.count.defaultOptions, customOptions);
        return this.each(function(){
            var $this = $(this);
            $this.text(options.startCount+'');
         var myInterval = window.setInterval(function(){
                var currentCount = parseFloat($this.text());
                var newCount = currentCount + 1;
                $this.text(newCount+'');
        }, 1000);
        });
    };


    $.fn.count.defaultOptions = {
```

```
    startCount:100
    };
})(jQuery);

// Overwrite default options
jQuery.fn.count.defaultOptions.startCount = 200;

jQuery('#counter1').count(); // Will use startCount: 200, instead of startCount:0
jQuery('#counter2').count({startCount:500}); // Will overwrite any default values

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

Create elements on the fly, invoke plugins programmatically

Depending up the nature of the plugin, it can be critical that a plugin be called both normally (via DOM elements and events) as well as programmatically. Consider a dialog plugin. There will be times that the modal/dialog will open based on user events. Other times, a dialog will need to open based on environmental or system events. In these situations, you can still invoke your plugin without any elements in the DOM by creating an element on the fly in order to invoke the plugin. In the code below, I am invoking the `dialog()` plugin on page load, by first creating an element to invoke my plugin.

JS Bin: http://jsbin.com/aqopi/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<a href="#" title="Hi">dialog, say Hi</a>
<a href="#" title="Bye">dialog, say Bye</a>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>

(function($) {
    $.fn.dialog = function(options){
        var text = this.attr('title') || this.text();
     alert(text);
    };
})(jQuery);

jQuery('a').click(function(){ // Invoked by user event
    $(this).dialog();
    return false;
});

$(window).load(function(){
    // Create DOM element to invoke the plugin
    jQuery("<a></a>").attr('title', 'I say Hi when invoked!').dialog(); // Run
```

```
immediately
});


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

Obviously, there could be a lot of variation of this pattern depending on the options, complexity, and functionality of the plugin. The take-away here should be that plugins can be called via existing DOM elements, as well as those created on the fly.

Providing callbacks and passing context

When authoring jQuery plugins, it is a good idea to provide callback functions as an option, and to pass these functions the context of **this** when the callback is invoked. This provides a vehicle for additional treatment to elements in a wrapper set. In the code below, we are passing a custom option to the **outAndInFade()** plugin method, that is a function, that should be called once the animation is complete. The callback function is being passed the value of **this** when it's being invoked. This allows us to then use the **this** value inside the function we defined. When the callback function is invoked the keyword this will refer to one of the DOM elements contained within the wrapper set.

```
<!DOCTYPE html>
<html lang="en">
<body>
<div>Out And In Fade</div>
<div>Out And In Fade</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>

(function($) {
    $.fn.outAndInFade = function(customOptions) {
        var options = $.extend({}, $.fn.outAndInFade.defaultOptions, customOptions ||
{});

    return this.each(function() {
            $(this).fadeOut().fadeIn('normal', function() { // Callback for fadeIn()
                // Call complete() function, pass it "this"
                if ($.isFunction(options.complete)){
options.complete.apply(this);
}
        });
        });
    };

    $.fn.outAndInFade.defaultOptions = {
        complete: null // No default function
```

```
        };
})(jQuery);


jQuery('div').outAndInFade({
// Change background-color of the element being animated on complete.
    // Note: "this" will refer to the DOM element in the wrapper set.
complete: function(){
        $(this).css('background', '#ff9');
    }
});


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

Chapter 9 - Performance best practices

Use the latest and greatest version of jQuery

The jQuery development team is extremely active and constantly improving the jQuery code with performance enhancements. Therefore, it is important that you are always leveraging the latest release of jQuery. I advise that as a jQuery developer, you always read the release notes and consider immediately upgrading to new releases that contain [significant performance enhancements](#).

Passing the jQuery function a context can improve query performance

You can actually speed things up quite a bit when you pass the jQuery function a context in which to perform its query. Doing so will reduce number of DOM elements that are traversed. In order to do this, you need to pass the jQuery function a second parameter, which is a reference to a single DOM element. This element is then used as the starting point for the DOM query.

JS Bin: http://jsbin.com/osuga/edit/#html

```html
<!DOCTYPE html>
<html lang="en">
<body>
<div>
    <div>
        <div id="context">
            <a href="#">jQuery</a>
        </div>
    </div>
</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($) {

    alert($('a', document.getElementById('context')).context.id); // Alerts "context"

    alert($('a', '#context').context.nodeName); // No speed gain. Alerts "document"

})(jQuery);
</script>
</body>
</html>
```

This concept can also be applied in situations when the value of `this` is available. By passing the value of `this` as the context to search within we avoid traversing the entire document.

JS Bin: http://jsbin.com/uteve/edit/#html

```html
<!DOCTYPE html>
<html lang="en">
<body>
<div>
    <div>
        <div id="context">
            <a href="#">jQuery</a>
        </div>
    </div>
</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($) {

    $('div#context').click(function(){
        alert($('a', this).context); // Alerts "object HTMLDivElement"
    }).click();

})(jQuery);
</script>
</body>
</html>
```

Grokking selector performance

Not all jQuery selectors are created equal, when it comes to performance. Combine this with the fact that not all browsers, JavaScript engines, operating systems, nor computer hardward are created equal — and being dogmatic about selector performance gets extremely difficult.

Consider that selector performance is directly dependent upon a combination of the following:

- Type of selector used.
- Complexity of the selector expression.
- Complexity and size of the DOM that is being traversed.

- Availability of a native javascript function mapped to the selector — e.g. `getElementsByTagName()`, `getElementById()`, `getElementsByClassName()`, `querySelectorAll()`.
- Browser, operating system, and computer hardware.

Because of these variables, selector performance should be based on some simple and loose rules:

- Select DOM elements using an ID selector whenever possible. It is fastest because all browsers support `getElementById()`. If you have to use a class name, do not use it alone. Use `$('td.class')` or `$('#nav .class')`. Beyond this, the selector types themselves provide only negligible speed improvements.
- Passing the jQuery function one selector is much faster than passing multiple selector expressions.
- In general, the simpler the selector expression, the better the performers: — e.g `$('#div div.class .right#fine')` vs. `$('#fine')`.
- Simple DOM structures are traversed faster than larger, more complex structures. Thus, context becomes very important when selecting elements within large DOM structures.

Equipped with this knowledge, it should not be difficult to master the balance between what selectors you pass the jQuery function, and the performance of the code. Selectors are even faster if they have native JavaScript equivalents that can be leveraged by jQuery — e.g. `getElementsByClassName()`.

> **Notes:**
>
> - In most cases providing a context for the jQuery function to traverse out-performs fine-tuning selectors

Cache sets of selected elements that are used more than once

Do not duplicate queries for the same element(s) when you already have a pointer to the query result. If you intend to reuse a wrapped set of element(s) that are already selected, cache the result of the selector in a local variable to leverage what is already in memory.

This concept is particularly important when dealing with loops. A jQuery wrapper set should be stored in a local variable outside the body of a loop, so that you avoid repeating the same query for each iteration.

```
// Slow example:
for (i = 0; i < $('ul li').length; i++) { // Runs $('ul li') every iteration
$('ul').eq(i).text(); // Runs $('ul') every iteration
}

// Fast example:
```

```
var list = $('ul'); // Store query
var listLength = list.find('li').length; // Store length
for (i = 0; i < listLength; i++) { // Query pulled from memory
    list.eq(i).text() // Query pulled from memory
}
```

Keep DOM changes to a minimum

It is critically important for web page optimization/performance that DOM changes are keep to a minimum. With this in mind, you should always approach DOM changes with the attitude of less is more. Less interaction with the DOM typically equates to a faster user interface.

This concept is most apparent when it comes to working with loops. In the code below the first loop is accessing/updating the DOM 100 times. This is simply unacceptable when all that is really needed is a single DOM interaction.

```
// Slow example:
var list = $('ul');
for (i = 0; i < 100; i++) {
list.append('<li>List item ' + i + '</li>'); // Interacts with the DOM 100 times
}

// Fast example:
var listItems = ''; // Empty string

for (i = 0; i < 100; i++) {
listItems += '<li>List item ' + i + '</li>'; // Single string containing DOM structure
}

$('ul').html(myListItems); // Use innerHTML to update the DOM only once
```

In the second loop we use a string to contain the DOM structure. Once this is done, the string is used to update the DOM only once by passing the `html()` method the string representation of the DOM structure.

> **Notes:**
>
> - To speed up `html()` forgo its use by using the innerHTML property of an HTML DOM element. Instead of `$('div').html(stringOfHTML)` do `$('div')[0].innerHTML = stringOfHTML`. By doing this, you bypass the overhead of using the `html()` method.

Optimize by passing jQuery methods a key/value object

The jQuery methods `attr()` and `css()` both accept an object of properties and values, as well as a single property/value pair. Passing an object increases the performance of the code by cutting down on the number of jQuery objects created and repetitive code run

```
// Slow example:
$('a').css('display', 'block');
$('a').css('color', 'red');
$('a').attr('title', 'Title Txt');
$('a').atrr('href', 'http://www.jquery.com');

// Fase example:
$('a')
    .css({'display': 'block', 'color': 'red'})
    .attr({'title': 'Title Txt', 'href': 'http://www.jquery.com'});
```

Optimize by passing multiple selectors to the jQuery function

It is possible to group selectors together and pass them to jQuery as a single parameter with selectors separated by commas (just like in CSS). Doing this increases the performance of the code by cutting down on the number of jQuery objects created and repetitive code run.

```
// Slow example:
$('#div1').hide();
$('#div2').hide();
$('#div3').hide();

// Fast example:
$('#div1, #div2, #div3').hide();
```

Optimize by leveraging chaining

Do not perform unnecessary DOM traversal. Use chaining when it makes sense and provides an increase in efficiency. Doing so increases the performance of the code by cutting down on needless repetition.

```
// Slow example, creates three instances of jQuery object
$('div .open').hide();
$('div .close').show();
$('div').fadeIn();

// Fast example, uses chaining to re-purpose the orginal wrapper set
$('div')
    .find('.open')
    .hide()
.end()
    .find('.close')
    .show()
.end()
    .fadeIn();

// Alternatively, do this:
var $div = $(div); // Cache the wrapper set
$div.find('.open').hide();
$div.find('.close').show();
$div.fadeIn();
```

Use the native `for` loop when dealing with big loops

Native browser functions are always faster! The question that needs to be answered in most cases is: How much faster? In most cases the increase is negligible. If that is the case, go ahead and use the jQuery `each()` method or utility function over a native JavaScript for loop.

The rule of thumb I follow is most small looping tasks (less than a 1000 iterations) are perfectly suited for the jQuery `each()` utility method. The abstracted jQuery solutions for looping allow for easier coding, which I value over the negligible speed increase won by using a native approach with smaller loops. Truth be told, if I cannot feel a significant difference when using a page, I do not worry about optimization.

Once you get into bigger loops (1000+ iterations) you might start measuring the difference by documenting the time it takes for a native function to execute vs. a jQuery solution. This test becomes critical when it comes to Internet Explorer, which runs notoriously slow with big loops of any kind.

Apply visual changes via ID and Class vs. manipulating style properties

When dealing with CSS properties it is always more efficient to update the DOM with a new `class` name or `id` than to update the actual `style` propereties of a DOM element via `.css()`.

```html
<!DOCTYPE html>
<html lang="en">
<style>
.newStyles {
    background-color: red;
    display:block;
    height:100px;
    width:100px;
}
</style>

<body>
<div></div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($) {

    // Slow example:
    $('div').css({
        'display': 'block',
        'width': '100px',
        'height': '100px',
        'background-color': '#f00'
    });

    // Fast example:
    $('div').addClass('newStyles');

})(jQuery);
</script>
</body>
</html>
```

Chapter 10 - Effects

Disable all jQuery effect methods

It is possible to disable all of the animating methods jQuery provides by simply setting the value of the **off** property to **true**.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<div style="height:100px; width:100px; background-color:red; position:absolute;
left:20px;">
    Try to animate me!
</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($) {

    jQuery.fx.off = true;
    $('div').slideUp(); // Does not animate, hides immediately

})(jQuery);
</script>
</body>
</html>
```

When **off** is set to **true** all the effect methods will not animate and will instead be hidden and shown immediately using the CSS rules **display:none** and **display:block**. You can turn the animation back on by passing the **off** property a **false** value.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<div style="height:100px; width:100px; background-color:red; position:absolute;
left:20px;">
    Try to animate me!
</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($) {

    jQuery.fx.off = true;
```

```
    $('div').slideUp();

    jQuery.fx.off = false; // Turn animation back on
    $('div').slideDown(); // Will now animate


})(jQuery);
</script>
</body>
</html>
```

Grokking the `stop()` animation method

It is often necessary to stop an animation currently in-progress before starting another. For example, when using the custom `mouseenter` and `mouseleave` events (or `hover()` method) you may unintentionally trigger an element that is already animating due to a previous `mouseenter` or `mouseleave` event. This causes a build-up of queued animations which, resulting in a sluggish interface. To avoid this, simply use the `stop()` method to stop the current animation before starting a new one.

JS Bin: http://jsbin.com/iriwu/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<div style="height:100px; width:100px; background-color:red; position:absolute;
left:20px;">
    Hover over Me!
</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($) {

    $('div').hover(
        function() {
            $(this).stop().animate({left: 75}, 'fast');
        },
        function() {
            $(this).stop().animate({left: 20}, 'fast');
        }
    );

})(jQuery);
</script>
</body>
</html>
```

Remove the `stop()` methods from the code, and roll the mouse over the element several times to see the ghost animations occur. Continuously rolling-over the element in the page will result in animation build up, which is typically not the desired result.

> **Notes:**
>
> - Additionally, it is possible to not only stop the current animation in the queue for the select element but also the entire queue by passing the `stop()` method a parameter of true. This will effectively stop *all* queued animations, active and inactive.

Determine if an element is animating using `:animated`

The custom `:animated` selector filter can be used to select elements that are currently animating. Below I use this custom selector filter to add text to an animating `<div>` element.

JS Bin: http://jsbin.com/uzebe/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<div style="height:100px; width:100px; background-color:red; color:white"></div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($) {

    function recursiveAnimate(){
        $('div').slideToggle('slow',recursiveAnimate);
    };

    recursiveAnimate();

    $('div:animated').text('I am animating');

})(jQuery);
</script>
</body>
</html>
```

Using `show()`, `hide()`, and `toggle()`, without animation

Using the `show()`, `hide()`, and `toggle()` methods with a parameter will cause the elements being shown or hidden to animate by changing CSS properties: height, width, opacity, margin, padding. It is possible to skip the the animations for hiding and showing elements, simply by not passing any parameters. This changes how these methods adjust the visibility of an element. Affected elements

will simply appear or disappear without any animation, by adjusting the CSS `display` property instead.

```html
<!DOCTYPE html>
<html lang="en">
<head>
<style type="text/css">
div {
    height: 100px;
    width: 100px;
    background-color: red;
    color: white;
    margin: 5px;
}
</style>
</head>
<body>
<div>Click Me (hide animation)</div>
<div>Click Me (hide no animation)</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($) {

    // Hide, with animation
    $('div:first').click(function(){$(this).hide(1000)});

    // Hide, no animation
    $('div:last').click(function(){$(this).hide()});

})(jQuery);
</script>
</body>
</html>
```

Notes:

- The jQuery methods `hide()`, `show()`, `toggle()`, `slideUp()`, `slideDown()`, `slideToggle()` when used on elements that have a CSS `display` value of `inline` will be changed to `display:block` for the duration of the animation.

Grokking sequential and non-sequential animations

It is important to understand the difference between animations that happen simultaneously, and animations that occur in a sequential order over a period of time. By default, when effect methods are chained, they are added to a queue and each effect occurs one after another.

```
<!DOCTYPE html>
<html lang="en">
<body>
<div style="height:100px; width:100px; background-color:red; position:absolute;
left:20px; border:1px solid #ff9933">
    Animate me!
</div>
<div style="height:100px; width:100px; background-color:red; position:absolute;
left:20px; top:100px; border:1px solid #ff9933">
    Animate me!
</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

    // Each effect is added to a queue and occurs sequentially
    $('div:first').slideUp('slow').slideDown('slow').hide('slow');

    // Each method is added to a queue and occurs sequentially
    $('div:last').animate({width:'200px'},1000).animate({borderLeftWidth:'10px'},1000);

})(jQuery);
</script>
</body>
</html>
```

Using the `animate()` method, we can set animations to occur non-sequential or at the same time by passing all the CSS property changes to a single `animate()` method call. In the code below the `<div>` will animate its width and border left width at the same time.

```
<!DOCTYPE html>
<html lang="en">
<body>
<div style="height:100px; width:100px; background-color:red; position:absolute;
left:20px; border:1px solid #ff9933">
    Animate me!
</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

    // Both width and borderLeftWidth properties will animate simultaneously
    $('div').animate({width:'200px',borderLeftWidth:'10px'},1000);

})(jQuery);
</script>
</body>
</html>
```

`Animate()` is the base low-level abstraction

The `animate()` method is the base method which is used to construct all the pre-configured animations — e.g. `hide()`, `slideDown()`. It provides the ability to change (over time) the values of style properties.

Here is what you need to know when using this method.

- Only properties that take numeric values are supported. In other words, you can't animate the value of say the `backgroundColor` property (at least not without a plugin). Also, properties that take more than one value like `backgroundPosition` can't be animated.
- As of jQuery 1.2, you can animate CSS properties by using em and % where applicable.
- Relative animations can be spedified using "+=" or "-=" in front of the property value. This would, for example, move an element positively or negatively realative to its current position.
- As of jQuery 1.3, if you specify an animation duration of 0, and the animation will immediately set the elements to their end state.
- As a shortcut, if a value of `toggle` is passed an animation will simply reverse from where it is at and animate to that end.
- All CSS properties set via a single `animate()` method will animate at the same time.

Grokking the jQuery fading methods

Three concepts need to be called out when using `fadeIn()`, `fadeOut()`, `fadeTo()` methods.

- Unlike other effect methods, fading methods only adjust the opacity of an element. It is assumed when using these effect methods that any element being faded already has a height and width.
- Fading animations will fade elements from their current opacity.
- Using the `fadeOut()` method will fade an element from its current opacity and then once 100% faded it will change the CSS display property of the element to "none".

Each of the aforementioned points are being illustrated in the code below.

JS Bin: http://jsbin.com/ojonu/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<!-- Elements being faded should have a width and height -->
<div style="height:100px; width:100px; background-color:red;"></div>
<button>Fade the rest of the way</button>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
```

```
<script>
(function($) {

    $('div').fadeTo('slow', 0.50);
    $('button').click(function(){
        // Fade from current opacity to zero,
        // then hide element via display:none
        $('div').fadeOut('slow');
    });

})(jQuery);
</script>
</body>
</html>
```

Chapter 11- AJAX

The jQuery `ajax()` function is the lowest-level abstraction

The jQuery `ajax()` function is the lowest level of abstraction available for doing [XMLHttpRequests](#) (aka [AJAX](#)). All the other jQuery AJAX functions, such as `load()`, leverage the `ajax()` function. Using the `ajax()` function provides the greatest functionality avaliable for doing [XMLHttpRequests](#). jQuery also provides other higher level abstractions for doing very specific types of [XMLHttpRequests](#). These functions are essentially shortcuts for the `ajax()` method.

These shortcut methods are:

- [load()](#)
- [get()](#)
- [getJSON()](#)
- [getScript()](#)
- [post()](#)

The take-away here is that while the shortcuts are nice at times, they all use `ajax()` behind the scenes. And, when you want all the features and customizations that jQuery offers when it comes to AJAX, then you should just use the `ajax()` method.

> **Notes:**
>
> - By default the `ajax()` and `load()` ajax functions both use the GET HTTP protocol.

jQuery supports cross-domain JSONP

JSON with Padding ([JSONP](#)) is a technique that allows the sender of an HTTP request, where JSON is returned, to provide a name for a function that is invoked with the JSON object as a parameter of the function. This technique does not use XHR. It uses the script element so data can be pulled into any site, from any site. The purpose is to circumvent the same-source policy limitations of XmlHttpRequest.

Using the `getJSON()` jQuery function we can load JSON data from another domain when a [JSONP](#)

callback function is added to the URL. As an example, here is what a URL request would look like using the flickr API.

```
http://api.flickr.com/services/feeds/
photos_public.gne?tags=resig&tagmode=all&format=json&jsoncallback=?
```

The **?** value is used as a shortcut that tells jQuery to call the function that is passed as a parameter of the **getJSON()** function. You could replace the **?** with the name of another function if you do not want to use this shortcut.

Below I am pulling into a web page, using the flickr JSONP API, the most recent photos tagged with "resig". Notice that I am using the **?** shortcut so jQuery will simply call the callback function I provided the **getJSON()** function. The parameter passed to the callback function is the JSON data requested.

JS Bin: http://jsbin.com/oheke/edit/#html

```
<!DOCTYPE html>
<html lang="en">
<body>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

$.getJSON("http://api.flickr.com/services/feeds/
photos_public.gne?tags=resig&tagmode=all&format=json&jsoncallback=?", // Using ? just
means call the callback function provided
function(data){ // Data is the JSON object from Flickr
    $.each(data.items, function(i,item){
        $('<img />').attr("src", item.media.m).appendTo('body');
        if ( i == 30 ) return false;
    });
});

})(jQuery);
</script>
</body>
</html>
```

Notes:

- Make sure you check the API of the service you are using for the correct usage of the callback. As an example, Flickr uses the name **jsoncallback=?** whereas Yahoo! and Digg use the name **callback=?**.

Stop a browser from caching XHR requests

When doing an XHR request, Internet Explorer will cache the response. If the response contains static content with a long shelf life, caching may be a good thing. However, if the content being requested is dynamic and could change by the second, you will want to make sure that the request is not cached by the browser. One possible solution is to append a unique query string value to the end of the URL. This will ensure that for each request the browser is requesting a unique URL.

```
// Add unique query string at end of the URL
jQuery.ajax(
 url:'some.php?nocache='+(new Date()).getTime(),
type:'POST'
);
```

Another solution, which is a more of a global solution, is to set up all Ajax requests by default to contain the no-cache logic we just discussed. To do this, use the `ajaxSetup` function to globally switch off caching.

```
$.ajaxSetup ({
cache: false // True by default. False means caching is off.
});
```

Now, in order to overwrite this global setting with individual XHR requests, you simply change the cache option when using the `ajax()` function. Below is a coded example of doing an XHR request using the `ajax()` function which will overwrite the global setting and cache the request.

```
$.ajaxSetup ({
cache: false // True by default. False means caching is off.
});

jQuery.ajax(
    cache: true,
    url: 'some.php',
    type:'POST'
);
```

Chaper 12 - Miscellaneous concepts

Storing data on DOM elements

Using the `data()` method, it is possible to store/associate JavaScript values (strings, numbers, booleans, objects, arrays) with DOM elements. The solution of storing data on DOM elements is preferred over storing information as a value of an element's attribute. For example, we should avoid storing data on the `title`, `alt`, or `rel` attributes. Below I am storing the color selected in the list on the `<ul>` element, which can then be accessed to retrieve the stored value.

```
                                                        JS Bin: Blows up on this one!
<!DOCTYPE html>
<html lang="en">
<style type="text/css">

.selected {
    background-color: #ffc;
}

</style>
<body>
<ul>
    <li><a href="#">red</a></li>
    <li><a href="#">orange</a></li>
    <li><a href="#">blue</a></li>
    <li><a href="#">green</a></li>
</ul>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

    $('a').click(function(){
        $this = $(this);
        $ul = $this.closest('ul');
        $ul.data('selectedColor', $this.text()); // Store text() of the <a>
        $ul.find('a').removeClass('selected').end().end().addClass('selected');
        alert($ul.data('selectedColor')); // Alert selected color
    })

})(jQuery);
</script>
</body>
</html>
```

115

While the `data()` method is typically used on an element selected via the jQuery function — e.g. `$('div').data()` — it is also possible to use the `data()` method as a stand-alone function.

The jQuery documentation details both usages:

- http://docs.jquery.com/Core/data
- http://docs.jquery.com/Internals/jQuery.data

When storing data in the DOM the `data()` feature from jQuery should be your go-to solution.

Adding new functions to the jQuery namespace

It is often useful to reuse the jQuery namespace (as opposed to creating your own) for your own function declarations so as to avoid creating global code that could potentially create conflicts. Additionally, it is recommended that any jQuery related plugin-like functions that do not require a set of DOM elements should be stored in the jQuery namespace. This can be done by simply adding a property to the jQuery object. In the code below we are adding a new function to the jQuery object called `customAlert()`.

```
JS Bin: http://jsbin.com/ixubi/edit/#html

<!DOCTYPE html>
<html lang="en">
<body>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

    $.customAlert = function(text){ // Store customAlert function in jQuery object
        alert(text);
    };

    $.customAlert('Hi'); // Invoke customAlert function

})(jQuery);
</script>
</body>
</html>
```

If you had several functions related to each other, you should house them in another namespace, to avoid clutter in the jQuery namespace. In the code below, we create a myDialog object, which is stored in the jQuery object. Inside `myDialog`, we house multiple functions that are related to `myDialog`.

```
$.myDialog = { // Create myDialog object
    show: function(){ // Show code },
    hide: function(){ // Hide code },
    position: function(){ // Position code },
    initiate: function(){ // Initiate code }
};


$.myDialog.initiate()
```

Computing an element's attribute value

Before setting the value of an element's attribute, it is possible to compute a value by passing the `attr()` method a function, instead of a string as the second parameter. This allows you to iterate over the elements in the wrapper set and add a unique attribute value on each iteration.

```
<!DOCTYPE html>
<html lang="en">
<body>
<div>0</div>
<div>1</div>
<div>2</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

    // Iterate over each <div>, giving each a unique id
    $('div').attr('id',function(i){ return 'div'+i});

// Could also be written:
/*
    $('div').each(function(i){
        $(this).attr('id', 'div' + i);
    });
*/

})(jQuery);
</script>
</body>
</html>
```

Should I use CSS properties or JavaScript references?

When getting or setting CSS properties on an HTML element, you have the option of either using the hyphenated string version of a CSS property name (`background-color`) or the JavaScript property reference (`backgroundColor`). The important thing to remember, regardless of which you choose, is that both have to be passed to the jQuery `css()` method within quotes. Personally, I like using the hyphenated property names because they match what you already write when using CSS styles.

JS Bin: http://jsbin.com/anizo/edit/#html

```html
<!DOCTYPE html>
<html lang="en">
<body>
<p style="background-color:#990033">jQuery</p>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

    alert($('p').css('background-color'));
    alert($('p').css('backgroundColor'));

})(jQuery);
</script>
</body>
</html>
```

CSS Property	JavaScript Reference
background	background
background-attachment	backgroundAttachment
background-color	backgroundColor
background-image	backgroundImage
background-position	backgroundPosition
background-repeat	backgroundRepeat
border	border
border-bottom	borderBottom
border-bottom-color	borderBottomColor
border-bottom-style	borderBottomStyle
border-bottom-width	borderBottomWidth
border-color	borderColor
border-left	borderLeft
border-left-color	borderLeftColor
border-left-style	borderLeftStyle
border-left-width	borderLeftWidth
border-right	borderRight
border-right-color	borderRightColor

border-right-style	borderRightStyle
border-right-width	borderRightWidth
border-style	borderStyle
border-top	borderTop
border-top-color	borderTopColor
border-top-style	borderTopStyle
border-top-width	borderTopWidth
border-width	borderWidth
clear	clear
clip	clip
color	color
cursor	cursor
display	display
filter	filter
font	font
font-family	fontFamily
font-size	fontSize
font-variant	fontVariant
font-weight	fontWeight
height	height
left	left
letter-spacing	letterSpacing
line-height	lineHeight
list-style	listStyle
list-style-image	listStyleImage
list-style-position	listStylePosition
list-style-type	listStyleType
margin	margin
margin-bottom	marginBottom
margin-left	marginLeft
margin-right	marginRight
margin-top	marginTop
overflow	overflow
padding	padding
padding-bottom	paddingBottom
padding-left	paddingLeft
padding-right	paddingRight
padding-top	paddingTop
page-break-after	pageBreakAfter
page-break-before	pageBreakBefore

position	position
float	styleFloat
text-align	textAlign
text-decoration	textDecoration
text-decoration: blink	textDecorationBlink
text-decoration: line-through	textDecorationLineThrough
text-decoration: none	textDecorationNone
text-decoration: overline	textDecorationOverline
text-decoration: underline	textDecorationUnderline
text-indent	textIndent
text-transform	textTransform
top	top
vertical-align	verticalAlign
visibility	visibility
width	width
z-index	zIndex

Accessing an iframe's content

When the `contents()` method is used on an iframe — e.g `$('#myiFrame').contents()` — containing a document from the same domain, it will retrieve the content document of the iframe, which gives us access to the iframe's DOM. This allows us to leverage jQuery to manipulate the iframe's DOM as if it were part of the parent page.

```
// Get innerHTML from the <body> in an iframe
$('iframe').contents().find('body').html();

// Set innerHTML of iframe's <body> element
$('iframe').contents().find('body').html('<p>Hi</p>');
```

Notes:

- Make sure you understand the cross-domain limitations of iframes before using this technique in a production environment.

Leverage a jQuery plugin for Flash embedding

I have been a long time fan of the [swfObject](#) script for embedding Flash files in HTML pages. However, the standalone swfObject script contains functionality that jQuery already handles. So, why not leverage jQuery to its fullest if you are already using it? Jonathan Neal had this same thought, and created a plugin called [jQuery-SWFObject.](#) Below I show this plugin in use on a simple HTML page.

```html
<!DOCTYPE html>
<html lang="en">
<body>
<div id="flash"></div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script src="http://jquery.thewikies.com/swfobject/jquery.swfobject.1-0-5.js"></script>
<script>
jQuery('#flash').flash({swf: 'flash.swf'}); // This should be a path to the *.swf
</script>
</body>
</html>
```

Pre-loading images

It is wise to pre-load images before the user needs them (for `:hover`). The following is a custom jQuery function for pre-loading images.

```javascript
(function($){

    // Define preloadImages()
    $.preloadImages = function(arrayOfImages){ // Add custom function to jQuery namespace
        $(arrayOfImages).each(function(){
            $('<img/>')[0].src = this; // Create img element, set src attribute
        });
    };

    // Use preloadImages()
    $(window).load(function(){ // Load images, after window.onload fires
    $.preloadImages(['img1.jpg', 'img2.jpg', 'img3.jpg']);
    });

})(jQuery);
```

This could also be written in the form of a plugin. See the chapter on plugins if this is new to you.

```
(function($){

    $.fn.preloadImages = function(){
        return this.each(function(){ // "this" is the jQuery object
            $('<img/>')[0].src = this; // "this" is the value in the array
        });
    };

    $(window).load(function(){
        $(['img1.jpg', 'img2.jpg', 'img3.jpg']).preloadImages(); // Pass jQuery an
array
    });

})(jQuery);
```

Pre-loading assets using XHR

It is possible to actually cache page assets using XHR HTTP requests. jQuery make this pretty simple. The concept is to request assets in the background (hidden from the user) after the page is done loading.

Using the `load()` event and the jQuery AJAX function `ajax()` we can request assets (.js, .gif, .png, .jpeg, .swf, .css) that we would like to pre-cache for the user. Below, I show a snippet of code that can be used to cache files from the server, via background processes once the page has been loaded.

```
(function($){

    $(window).load(function(){ // Wait for window.onload
     // Then, pre-load assets into cache
        $.ajax({ url:"javascript.js", dataType:"text" });
        $.ajax({ url:"image.gif", dataType:"text" });
        $.ajax({ url:"flash.swf", dataType:"text" });
        $.ajax({ url:"styles.css", dataType:"text" });
    });

})(jQuery);
```

Notes:

- Keep in mind that these files are being cached by default. But you could pass the `ajax()` function a parameter (`{cache: false}`) that tells the HTTP request not to cache the requested URL. It is possible to force the browser *not* to cache the assets you request. This might seem pointless since we are trying to cache the files, but it is mentioned for those situations where you want the user to get a freshly cached version only once.

Add a `class` to the html element as a CSS hook for JavaScript enabled browsers

As a fail-safe solution, we can verify that the user's browser is able to run JavaScript by placing a class attribute value on the HTML element to be used as a CSS hook. Doing so gives us a class name that we can use to target JavaScript enabled browsers with specific CSS properties. This becomes extremely handy when we would like portions of the interface hidden on page load only if the browser has JavaScript enabled. If the JavaScript is not enabled, then no HTML elements are hidden. In the following example, all <div> elements are hidden, but only if JavaScript is enabled.

JS Bin: http://jsbin.com/udova/edit/#html

```html
<!DOCTYPE html>
<html lang="en">
<style>

.js div {
    display: none;
}

</style>
<body>
<div>Hide if JavaScript is enabled!</div>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"></script>
<script>
(function($){

    $('html').addClass('js');

})(jQuery);
</script>
</body>
</html>
```