Working with jQuery, Part 1: Bringing desktop applications to the browser

Core functions, selection, and traversal of results

Skill Level: Intermediate

Michael Abernethy (mike@abernethysoft.com)
Product Development Manager
Optimal Auctions

09 Sep 2008

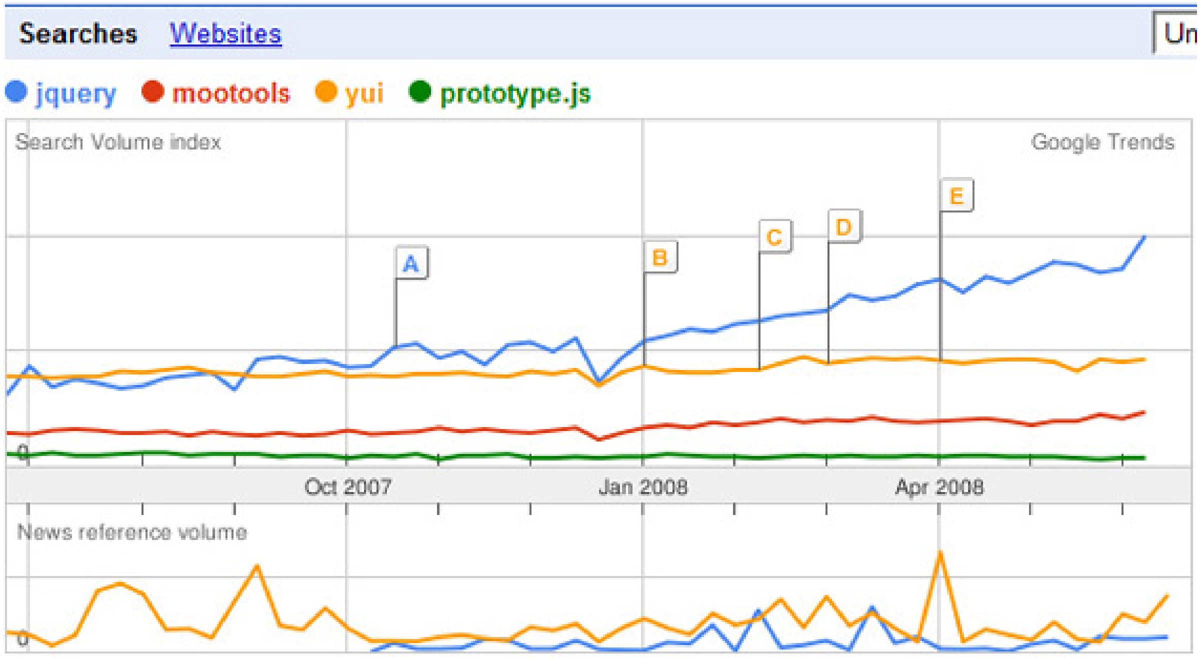
jQuery is emerging as the JavaScript library of choice for developers looking to ease their creation of dynamic Rich Internet Applications. As browser-based applications continue to replace desktop applications, the use of these libraries will only continue to grow. Get to know jQuery in this series of articles and learn how you can implement it in your own Web application projects.

Introduction

jQuery has distanced itself from other JavaScript library options to become the library of choice for Web developers and is fast becoming the first choice for programmers looking to ease their client-side development and create Rich Internet Applications (RIA) quickly and efficiently. As the use of RIA becomes ever-more prevalent in the world, the use of JavaScript libraries to assist in their development will continue to increase as well. RIAs are defined (loosely) as applications run through the browser that use a combination of CSS/JavaScript/Ajax to create the appearance of working on a desktop application. The latest features being added to recent releases of Firefox, Internet Explorer, Safari, and the recent release of Google's new Chrome browser, are focused on beefing up the speed of each browser's internal JavaScript engine for the sole purpose of making it more conducive to the type of RIAs that the browser makers picture us using in the near future. These companies envision Web pages that contain tens of thousands of lines of JavaScript code, making the importance of a mature and bug-free library from which to start all the more crucial.

So, as the future of Web applications moves toward these rich and immersive interfaces, Web developers increasingly turn to tools to ease that work. There are several JavaScript libraries out there in the world right now, each with its own strengths and weaknesses and its own fanboys and critics. I'm not here to debate which one is better than the other as far as features, because ultimately it doesn't really matter. Ultimately, what matters most is what library is getting used more than the others—volume matters. Take a look at the Google Trends graph of the four most popular JavaScript libraries below. It is apparent that over the past six to eight months, jQuery has become the dominant choice for JavaScript libraries and is growing rapidly.

Figure 1. Google Trends of common JavaScript libraries



The job market is also showing the emergence of jQuery as the JavaScript library of choice. A non-scientific look at Monster.com shows there are 113 jobs with "jQuery" listed, and only 67, 19, and 13 jobs listed for YUI, ExtJS, and mootools, respectively.

This first article in this jQuery series begins by exploring the syntax of jQuery, how it is set up, and how its functions are called. Later sections in this article explore the core functions in the library and how it uses its powerful selectors and filters to make DOM traversal easy and straightforward. Later articles will address CSS manipulation, form control, text changes, Ajax simplicity, and animations (everyone's favorite eye candy). One of the most interesting features of jQuery is its plug-in architecture, allowing many developers to add on to the functionality of jQuery. The final article will introduce you to many of the powerful plug-ins available to complete your RIA development process.

This series of articles is intended for people who have prior knowledge of JavaScript syntax, CSS syntax, and DOM syntax. If you need a refresher on this syntax before reading the articles in this series, I highly recommend the W3Schools links in the Resources section of this article.

The basics

Before getting into the fun stuff with jQuery, we need to get the basics out of the way—how to install it, get it started, and so on. Start by downloading the jQuery library provided in the Downloads section, and link to it like you would any other external JavaScript file:

Listing 1. How to install jQuery in your code

```
<script type="text/javascript" src="jquery.js"></script>
```

Because jQuery calls or manipulates the DOM objects, you would run into problems if you were manipulating these objects in JavaScript code immediately, before the document finished loading all the elements on the page. Conversely, you also don't want to have to wait until *everything* on the page has loaded—all the images, banner ads, analytic code, and YouTube video previews— before you can call jQuery code. Appropriately, a middle ground allows you to call jQuery code in a safe and error-free manner once the document has finished loading all the elements on the page, but before all the image, linking, and rendering is complete. To stress this again in a different way, all of your jQuery code needs to be in this function on a page, or in its own function. Do not put jQuery code in a JavaScript code section if it's not in a function.

Listing 2. How to properly call jQuery functions

```
// Incorrect
<script language=JavaScript>
   $("div").addClass("a");
</script>

// Correct
$(document).ready(function(){
   $("div").addClass("a");
 });

// - or -

$(document).ready(function(){
   myAddClass();
 });

function myAddClass()
{
   $("div").addClass("a");
}
```

Also, one additional helpful note: You can have as many document.ready() functions as you need on one page, and they will be called in succession. This is a good thing to keep in mind if you are building your pages dynamically with modules, and each module has its own supporting jQuery code (for example, a PHP page constructed of many smaller PHP page snippets).

One of the most interesting features of jQuery is its "chainability", its ability to put together a series of functions in order to improve readability and ease coding. Almost every jQuery function returns a jQuery object, meaning you can simply call additional functions on it over and over again to chain together a complete jQuery command. I compare this to the String class in Java, in which several functions return a String object, allowing you to chain together multiple functions on one line:

Listing 3. jQuery chainability

```
String man = new String("manipulated").toUpperCase().substring(0,5).toLowerCase();

$("div").addClass("a").show().text("manipulated");
```

Finally, the last thing to remember when working with jQuery, or any JavaScript library, is that they don't always work well with each other. In other words, when working with two or more libraries, the variable "$" is used by more than one library, meaning the engine won't know which library should be referenced with a "$" call. A perfect example of this is the CakePHP library, which includes prototype.js built in. Attempting to use jQuery on these pages will result in errors unless corrected. To get around this problem, jQuery provides a way to map the "$" variable to a different variable, for example:

Listing 4. jQuery conflict resolution

```
j$ = jQuery.noConflict();
j$("div").addClass("a");
```

Selection

The root of all jQuery is its ability to select certain elements on a page and manipulate them. In a sense, these are the objects around which the jQuery library is built to function. So, with the numerous options available on a normal HTML page, you need a way to quickly and efficiently select the elements you wish to work with on the page, selecting only those you want (no more and no less). Expectedly, jQuery provides powerful selection methods that allow you to find and select objects on the page. jQuery has created its own syntax for selection, and it is quite easy to learn.

(Many of the examples below use functions I won't get into until the next article, but they should be straightforward enough to understand what they are trying to do.)

At its root, the selection process in jQuery is really a giant filter process, whereby every element on the page is put through the filter you supply in your command, and it returns either the single matching object itself, or an Array of matching objects, from which you can traverse.

The first three examples are probably the most commonly used. They are finding objects by HTML tag, by ID, or by CLASS.

HTML

To get an array of all the matching HTML elements in a page, you can simply pass the HTML tag itself, without the braces, into the jQuery search field. This is the "quick and dirty" way of finding objects and is useful for attaching attributes to generic HTML elements.

Listing 5. HTML selection

```
// This will show every <div> tag in the page.  Note that it will show
// every <div>, not the first matching, or the last matching.
// Traversing Arrays is discussed later in the article.
$("div").show();

// This will give a red background to every <p> tag in the page.
$("p").css("background", "#ff0000");
```

ID

Proper page design calls for every ID on a page to be unique, though this is sometimes broken (intentionally or unintentionally). jQuery only returns the first matching element when using the ID selection, because it expects you to follow proper page design. If you need to attach a tag to several elements on the same page, the CLASS tag is the proper choice.

Listing 6. ID selection

```
// This will set the innerHTML of a span element with the id of "sampleText" to "Hi".
// Note the initial "#" in the command.  This is the syntax used by jQuery to search
// for IDs, and must be included.  If it is excluded, jQuery will search for the HTML
// tag instead, and with no <sampleText> tags on a page, will ultimately do
// nothing, leading to frustrating and hard-to-find bugs (not that that has ever
// happened to me of course).

$("#sampleText").html("Hi");
```

CLASS

Classes are a lot like IDs, except that they can be used for one or many elements on a page. Thus, while you are limited by having only one element per ID on a page, you can have many elements with the same CLASS on a page. This results in giving you the leeway to perform functions across wide-ranging elements on a page while passing in only one CLASS name.

Listing 7. CLASS selection

```
// This will create a red background on every element on the page with a CLASS of
// "redBack".  Notice that it doesn't matter which HTML element this "redBack"
// CLASS tag is attached to.  Also notice the period in the front of the query
//  term -- this is the jQuery syntax for finding the CLASS names.

$(".redBack").css("background", "#ff0000");

<p class="redBack">This is a paragraph</p>
<div class="redBack">This is a big div</div>
<table class="redBack"><tr><td>Sample table</td></tr></table>
```

Combining search criteria

The above three search criteria, and all of the filters presented below, can be combined in a search. By separating search criteria with a ",", the search will return a union of all matches of the search terms.

Listing 8. Combining searches

```
// This will hide every <p>, <span>, or <div>.
$("p, span, div").hide();
```

Further filters

While those are certainly the three most commonly used search parameters in jQuery, there are many others that can help you quickly find the elements you are looking for on a page. These further filters all start with a ":" to denote that they are the filters in the jQuery search term. Though they can stand alone as search criteria, they are designed primarily to be used with the three search criteria provided above, to allow you to fine tune search criteria to find the specific element you are looking for.

Listing 9. Further filters

```
// This will hide every <p> tag on a page
$("p").hide();

// This will hide the first element on a page, no matter its HTML tag
$(":first").hide();

// Notice how these can be used in combination to provide more fine tuning of
// search criteria.  This will hide only the first <p> tag on a given page.
```

```
$("p:first").hide();
```

Multiple filters can be used as search elements, and though I won't list them all here (that's what an API page is for after all), some of them are very handy in working with pages and searching for elements.

I *will* spend more time on some very important filters in the Selection package, and that is the *form* element filters. Rich Internet Applications today seem to focus on the form and its contained elements (text fields, buttons, check boxes, radio buttons, and so on) that gather and transmit information to and from the server. Because of their importance in RIAs, these filters have a particular importance in working with jQuery in today's types of Web applications.

These filters work like the other filters introduced in this article and are preceded with a ":" character to denote they are a filter. Also, they can be used in combination with other search filters to provide more refining. So, a search filter of ":text" will return every text field on a page, while a search filter of ".largeFont:text" will return only the text fields on a page that are part of the "largeFont" class. This will allow for further refinement and manipulation of form elements.

The form filters also include individual attributes of the elements as well, things that might be appropriate and good to know for developers. So, things like ":checked", ":disabled", and ":selected" will further refine the search criteria for a given search.

Traversal

Now that you've learned how to search and filter all of the elements on a page, you need an efficient way to traverse over the results and take actions on these elements. Not surprisingly, jQuery offers several ways of traversing over the results of these searches.

The first and most commonly used traversal technique is the each() function. This is programmatically the same as a "for loop", looping through each of the elements and incrementing the element with each iteration. Additionally, a reference to each element in the loop can be made with "this" (for using regular JavaScript syntax) or $(this) (for use in jQuery commands).

Take a look at the following example.

Listing 10. Each loop

```
// Will loop through each <p> tag on the page.  Notice the use of the
// inline function here -- this is analogous with the anonymous classes in Java.
// You can either call a separate function, or write an inline function like this.

var increment = 1;
```

```
$("p").each(function(){

    // now add a paragraph count in front of each of them.  Notice how we use the
    // $(this) variable to reference each of the paragraph elements individually.

    $(this).text(increment + ". " + $(this).text());
    increment++;
});
```

Since the search results are being stored in an array, you'd expect functions to work through the array like you would any data object in any programming language. Thus, to find the length of a given search result, you can call $().length on the array. Further array traversal functions are as follows in Listing 11 and fit appropriately with array traversal in other programming languages.

Listing 11. Additional array functions

```


// the eq() function lets you reference an element in the array directly.
// In this case, it will get the 3rd paragraph (0 referenced of course) and hide it
$("p").eq(2).hide();

// The slice() function lets you input a start and an end index in the array, to
// create a subset of the array.  This will hide the 3rd through 5th paragraphs on the
// page
$("p").slice(2,5).hide();
```

In addition to these array traversal functions, jQuery also offers functions that let you find elements nested right around your search terms. Why might this be useful? Well, oftentimes you want to embed a text label next to a picture, or an error message next to a form element. Using these commands lets you search for a specific form element and then place an error message directly next to it by placing it in the next element, a span tag. Listing 12 is an example of this design:

Listing 12. Example of next() function

```


<input type=text class=validate><span></span>

function validateForm()
{
    $(".validate:text").each(function(){
    if ($(this).val()=="")
    //  We'll loop through each textfield on the page with a class of "validate"
    //  and if they are blank, we will put text in the <span> immediately afterwards
    //  with the error message.

        $(this).next().html("This field cannot be blank");
});
}
```

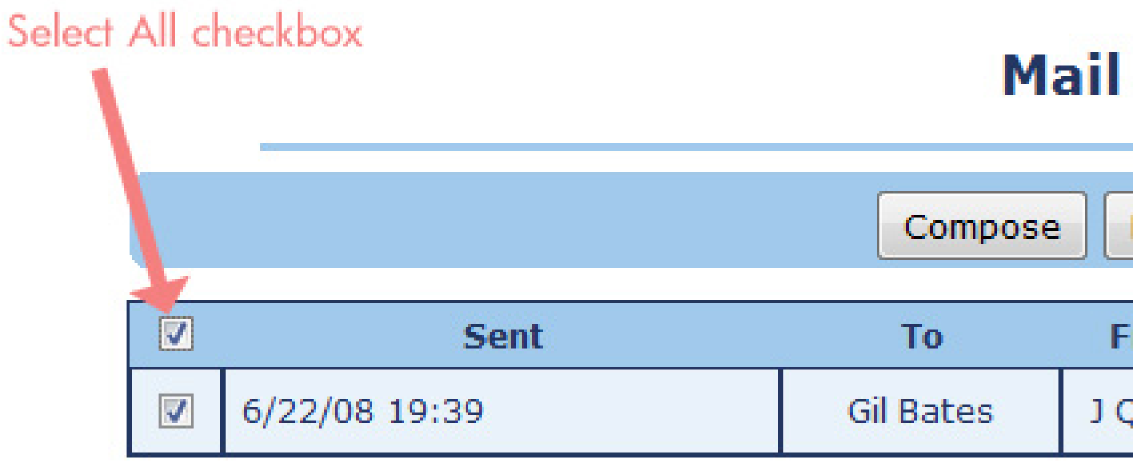
Bringing the lessons all together

To see how all of these lessons come together, take a look at the demo application included in this article. (See the Downloads section.)

A quick primer on the demo application is probably in order here. I'll be using the demo application throughout the article series, as it utilizes a number of different jQuery examples, and is an application that nearly everyone is familiar with—a Rich Internet Application for Web mail! This demo app is a simple mail client, utilizing jQuery to give users the feeling of working with an e-mail client that is actually a desktop application. By the end of the final article, you'll see how the simple application creates that look and feel for users, and ideally you'll see how easy it is to create this with jQuery.

This article focuses on the "Select All"/"Deselect All" check box that appears at the top left column in Web mail tables (highlighted below). When this check box is selected, it selects every check box in the column, and when it's deselected, it deselects every check box in the column.

Figure 2. "Select All" check box



Listing 13. Bringing the lessons together

```
<!-- The first step is creating the Select All checkbox itself.
we give it a unique ID on the page -->

<input type=checkbox id=selectall>

<!-- The next step is giving each of the rows their own checkbox.
we put each row's checkbox into the 'selectable' class, since there can be many rows,
and we want each of the rows' checkboxes to have the same behavior. -->

<input type=checkbox class=selectable>

<!-- The final step is bringing it all together with some jQuery code. -->

// remember that all jQuery setup code must be in this document.ready() function,
// or contained within its own function in order to function correctly.

$(document).ready(function(){
   // We use the jQuery selection syntax to find the selectall checkbox on the page
   // (note the '#' which signifies ID), and we tell jQuery to call the selectAll()
   // function every time someone clicks on the checkbox (we'll get to Events in a
   // future article).

   $("#selectall").click(selectAll);
});

// This function will get called every time someone clicks on the selectall checkbox
function selectAll()
{
    // this line determines if the selectall checkbox is checked or not.  The attr()
    // function, discussed in a future article, simply returns an attribute on the
    // given object.  In this case, it returns a boolean if true, or an undefined if
    // it's not checked.

    var checked = $("#selectall").attr("checked");

    // Now we use the jQuery selection syntax to find all the checkboxes on the page
    // with the selectable class added to them (each row's checkbox).  We get an array
    // of results back from this selection, and we can iterate through them using the
    // each() function, letting us work with each result one at a time.  Inside the
    // each() function, we can use the $(this) variable to reference each individual
    // result.  Thus, inside each loop, it finds the value of each checkbox and matches
    // it to the selectall checkbox.

    $(".selectable").each(function(){
       var subChecked = $(this).attr("checked");
       if (subChecked != checked)
          $(this).click();
    });
}
```

Conclusion

jQuery is becoming the preferred JavaScript library in the Web application development community, and it likely will continue to grow in importance as Rich Internet Applications become more and more prevalent. As multiple companies migrate their internal applications online, and as companies move their everyday desktop applications online (including word processors and spreadsheets), JavaScript libraries that ease development and promise cross-platform support will become part of the technology choice when an application is architected.

This first article in the series on jQuery introduced you to the jQuery syntax, how to use jQuery correctly in your own JavaScript code, and how to avoid any pitfalls when using it with other libraries. Further, it introduced you to the jQuery search and selection syntax, from which all the other functionality in jQuery is based. It allows you to find any page element you want, simply and quickly, and take actions on them. The article also showed you how to navigate through the results from these searches, letting you take actions on the elements individually. These two aspects of jQuery will be the fundamental basis on which the next articles in the series will build and on which all of your jQuery code will be based.

Finally, the demonstration application was introduced, a rich client Web mail application. In this article, you created the Select All/Deselect All check box by using the jQuery you learned in the article, and you saw that with only a few lines of code, you could create a common widget seen in many Web sites.

The next article in the series will add some interactivity to our sample Web application. You'll learn how to handle page events (clicking on elements, button clicks, combobox choices, and so on), how to get values from elements on the page, and how to alter the standard CSS on a page to change colors, layouts, and so on, without having to reload the page.

Downloads

Description	Name	Size	Download method
Zip file containing the sample application	jquery.zip	68KB	HTTP
War file containing the sample application	jquery.war	68KB	HTTP

Information about download methods

Resources

Learn

- Download the 1.2.6 Minimized jQuery which was the latest stable version at the time of this writing, and drop it in your own code.
- Read the complete jQuery API page to see all of the available functions in the library.
- Worried about jQuery's cross-browser compatibility? Check the list here for cross-browser support.
- Check out really cool advanced effects that you can create with jQuery.
- Get a thorough and complete background on CSS, JavaScript, and any other Web language at W3Schools.
- Get a general overview of jQuery in this introduction article "Simplify Ajax development with jQuery", (Jesse Skinner, developerWorks, April 2007).
- "Ajax overhaul, Part 2: Retrofit existing sites with jQuery, Ajax, tooltips, and lightboxes" (Brian Dillard, developerWorks, May 2008) is a look at some of the plugins available for jQuery, and how they expand the usefulness of the library.
- Expand your Web development skills with articles and tutorials that specialize in Web technologies in the developerWorks Web development zone.

Discuss

- Participate in developerWorks blogs and get involved in the developerWorks community.

About the author

Michael Abernethy
In his 10 years in technology, Michael Abernethy has worked with a wide variety of technologies and a wide variety of clients. He currently works as the Product Development Manager for Optimal Auctions, an auction software company. His focus nowadays is on Rich Internet Applications and making them both more complex and simpler at the same time. When he's not working at his computer, he can be found on the beach in Mexico with a good book.