

Intermediate JQuery

Performance metrics and tuning

Skill Level: Intermediate

[Michael Abernethy \(mabernethy@yahoo.com\)](mailto:mabernethy@yahoo.com)

Product Development Manager

Optimal Auctions

16 Jun 2009

jQuery is a great JavaScript library, but what about its performance? Is the trade-off between ease of use and a performance hit on the Web page worth it? Is there even a performance hit at all? This article answers your jQuery performance questions and offers some tips to improve its performance in your own applications.

Introduction

This article was prompted by a feedback e-mail I received from an earlier article; the e-mail said "jQuery is nice on a simple page, but on a complex page, the performance is terrible. Until the performance issues are fixed, you have to use regular JavaScript for a complex page." This immediately made me think "how does the performance of jQuery compare to JavaScript?" In fact, I have seen very few performance articles at all about the jQuery library compared to JavaScript, or compared to other JavaScript libraries. Maybe like most people, I saw how easy it was to work with client-side coding with jQuery, and I ignored any possible performance hits. To me, the trade-off would definitely be worth it. But, is there even a trade-off to consider? Is jQuery slower than "regular" JavaScript (whatever that is)? Is jQuery slower than other libraries? This article tries to answer these questions.

One of the most important things to consider when measuring JavaScript performance is the environment on which it is executing. Because the code is run on the client, it is dependent on the client's computer for execution, making the client machine the most important factor in performance. Obviously, the newest server

running a quad-core is going to execute code faster than an old 400 MHz processor. That goes without saying. However, because you cannot really control the machine that your Web app's users will use to visit your site, there's not a lot of reason to consider the hardware when measuring performance.

Another very important aspect of JavaScript performance, one that might not be evident to beginner JavaScript coders, is that the browser used to run the JavaScript is a very big factor. "Under the hood" of every browser is a JavaScript engine, native code that interprets and executes the JavaScript code you have written and takes actions on the pages of the Web application. The performance, therefore, is very dependent on the browser used by the clients, and in some situations, you *can* control what browsers your users use. This article gives you some guidelines on JavaScript performance and how it differs on different browsers.

Finally, after I've drawn some conclusions about the performance of JavaScript, and jQuery in particular, I will define some best practices for improving the performance of your jQuery code, taking advantage of the parts that I have found to be the fastest, and trying to avoid the parts that I've addressed as slow. By the time you have finished reading this article, your question of "Is the performance of jQuery good?" should be answered, and the e-mailer from above should know whether his claims are warranted.

Creating a performance test

The first step in performance testing is creating a suitable performance test. One of the biggest roles that jQuery, and any JavaScript library, plays in your coding is the use of selection to find specific page elements. It is this aspect that I will use in my initial performance testing. In my situation here, a good performance test would really put the JavaScript library through its paces, testing it against a page with thousands of page elements, hundreds of each kind. It would run all of the selection methods, so that I can see which selection methods are fast, and which ones are slow. It would know the right answers ahead of time, so it could determine if the JavaScript library was executing its selection correctly. Finally, it would present all of the results with appropriate times, allowing me to easily compare the results of all the libraries against each other.

Oh, and I forgot the most important aspect of this performance test: it should already be freely available. After all, this entire series has had one undertone to it, and that's to take advantage of the great work of others, so I will continue that theme here and use a JavaScript library performance test that has already been created. The test is called the SlickSpeed Selectors Test (see [Resources](#)) and it fits my needs extremely well. It compares jQuery 1.2.6 (the most recent version at the time this article was written) against 4 other popular JavaScript libraries (MooTools, Prototype, YUI, and Dojo). Then it runs 40 selection tests against a page with thousands of page elements. In other words, it's the perfect performance test for what I was looking for.

I'll be using this test for my initial performance test analysis.

Comparing performance of JavaScript libraries

For this initial performance test, I will be running it on a 2.2 GHz processor, 2 GB RAM, and (most importantly), Firefox 3.0.3. I ran the test 5 times on this setup, and Figure 1 shows the average results of the 5 runs.

Figure 1. Performance test 1 results

Test 1: Firefox 3.0.3 (all times in ms)					
	Mootools	jQuery	Prototype	YUI	Dojo
Run 1	425	456	691	1017	311
Run 2	412	456	728	995	295
Run 3	414	450	698	994	296
Run 4	418	448	698	1009	311
Run 5	422	447	692	1017	301
Average	418	451	701	1,006	303
Slowest - Fastest		704			
Slowest/Fastest		332%			

What conclusions can you draw from this initial test? Let's focus only on the overall results, and not on the individual tests that are run in the test at this point. I will look into each individual test a little later in this article, after I have drawn some broader conclusions.

- **Conclusion 1: Wow, YUI is really slow!**
Yes, YUI is particularly slow compared to other libraries. Looking at the individual tests for an explanation why shows that this library is very weak when it comes to selecting groups of elements (for example, "p, a"). This would be a very poor choice of a library for a complex page that depends on performance.
- **Conclusion 2: Mootools, jQuery, and Dojo are pretty much the same.**
These three libraries are all quite fast compared to the other two choices. Yes, Dojo was the fastest of the three. Yes, jQuery was the slowest of the three. But, they all are relatively of the same speed all things being considered.

- **Conclusion 3:** The relative difference between libraries is somewhat significant.
Measuring the slowest time/fastest time to determine a relative difference in speed, you can see that there's a 332% difference. This is significant, and it shows that your choice of JavaScript library can have an impact on your performance on the Firefox browser.

Keep in mind, though, that these conclusions are a result of working with only one browser. Let's take the conclusions from working with just Firefox 3.0.3 and move on to the next section, where I will run this test on different browsers.

JavaScript performance in different browsers

Many beginning Web programmers are surprised to find that each browser runs JavaScript differently, with different performance times and different results. While this can be frustrating to beginners, who are dismayed that they have to write extra code to handle different browsers, veteran Web programmers have been accustomed to this problem since the early days of Netscape versus Internet Explorer. This is also one of the main selling points of working with JavaScript libraries, because they take care of many or most of the browser differences.

The core reason for the difference in JavaScript speed is that each browser uses its own JavaScript engine. The engine is the native code that interprets the JavaScript and executes it against the Web application. So, as a result, the speed of JavaScript execution is directly attributable to the underlying engine and how much effort was put into performance with that engine. In the past few months, there has been definitive growth in the number of browser companies talking about their JavaScript engine performance, and with good reason. As the complexity of JavaScript grows on certain pages, the difference between a fast JavaScript engine and a slow JavaScript engine can make the difference between a Web application being responsive or slow and plodding. So, when companies like Google and Firefox discuss their JavaScript engines, and talk about the next generation being 10x faster than the current generation, this is important for Web applications, as the speed of the underlying JavaScript engine will lead directly to the growth of more complex and involved Web applications.

Because you now know that the JavaScript engine can be a factor in the speed of JavaScript execution, let's run the same test I ran on Firefox on several different browsers, attempting to quantify the effect that different engines can have on JavaScript performance. Keep in mind that these are the same tests I ran on Firefox, on the same laptop, so all things are equal except the JavaScript engine. Figure 2 shows the results.

Figure 2. Performance test 2 results

Test 2: Chrome 1.0 (all times in ms)					
	Mootools	jQuery	Prototype	YUI	Dojo
Run 1	202	167	883	724	208
Run 2	199	169	871	750	200
Average	201	168	877	737	204
Slowest - Fastest		709			
Slowest/Fastest		522%			

Test 3: IE 7.0.6 (all times in ms)					
	Mootools	jQuery	Prototype	YUI	Dojo
Run 1	1452	1091	6884	3693	1816
Run 2	1521	1104	6937	3813	2241
Average	1,487	1,098	6,911	3,753	2,029
Slowest - Fastest		5,813			
Slowest/Fastest		630%			

Test 4: IE 6.0.3790 (all times in ms)					
	Mootools	jQuery	Prototype	YUI	Dojo
Run 1	3127	1879	9953	9088	2691
Run 2	2049	1577	8484	7376	2292
Average	2,588	1,728	9,219	8,232	2,492
Slowest - Fastest		7,491			
Slowest/Fastest		534%			

Test 5: Safari 3.1.1 (all times in ms)					
	Mootools	jQuery	Prototype	YUI	Dojo
Run 1	400	561	907	2068	566
Run 2	391	580	706	2309	548
Average	396	571	807	2,189	557
Slowest - Fastest		1,793			
Slowest/Fastest		553%			

The first thing that you should notice when looking at these results is the wide range of times from the browsers. Focusing exclusively on jQuery for now, on Chrome 1.0, the jQuery test completed in 168 ms while it took 1728 ms on IE6. That's an incredible difference in times! jQuery selection can take over 10x longer on IE6 than it would on Chrome. You can see why Google brags about the speed of their JavaScript engine, and also why you don't hear much about some browsers' JavaScript engines. The differences are quite significant.

The other thing that should jump out at you is that jQuery, which was the 3rd fastest library in Firefox, is the fastest on some browsers and 3rd fastest on others. In fact, these results show that there are two groups of libraries in terms of performance, no matter the browser. Mootools, Dojo, and jQuery are always grouped together, and this group is significantly faster than the other group, which includes Prototype and YUI.

Conclusions from performance tests

I felt these conclusions deserved their own section, because they are very important for the JavaScript developer. Again, I'm attempting to draw conclusions from the above performance results, and will not be biased by the fact that these articles are about jQuery. I hope.

Conclusion 1: Mootools, jQuery, Dojo are pretty much the same performance-wise.

Looking at the results across all five browsers, you can see that when the results are normalized, these three libraries are nearly identical in terms of performance. (Ideally, I should have weighted the normalizations for each browser's market share. However, it would be tough to adjust these numbers, because sites that tend to have JavaScript libraries in use aren't typically those visited by the "average joe.")

Figure 3. Normalized test results (Mootools, jQuery, Dojo)

Normalized Results			
	Mootools	jQuery	Dojo
FF 3	1.38	1.49	1
IE 6	1.50	1	1.44
IE 7	1.35	1	1.85
Chrome	1.20	1	1.21
Safari	1	1.44	1.38
Average	1.29	1.19	1.38

Conclusion 2: Prototype and YUI are much slower in performance.

Looking at the results of these two libraries in comparison to jQuery in the 5 browsers, when the results are normalized you can see just how much of a difference in performance you will receive from these two libraries. On average, they run 300% slower than jQuery would in a browser.

Figure 4. Normalized test results (jQuery, Prototype, YUI)

Normalized Results			
	jQuery	Prototype	YUI
FF 3	1	1.55	2.23
IE 6	1	5.34	4.76
IE 7	1	6.29	3.42
Chrome	1	5.22	4.39
Safari	1	1.41	3.83
Average	1.00	3.96	3.73

Conclusion 3: If performance matters, the choice between Mootools, jQuery, and Dojo doesn't matter.

Based on the above normalized results, choosing one of these libraries will not give you a performance advantage over the other two choices. When taking all of the browsers into consideration, they all show pretty much the same performance. So, when you choose a JavaScript library, you can't go wrong by picking one of these three libraries.

Conclusion 4: If performance matters, don't choose Prototype or YUI.

If performance is a consideration when working with your JavaScript library, or you plan to create a large-scale JavaScript project, you probably should not choose one of these two libraries. The performance hit from choosing one of these libraries is significant in comparison to other choices.

Conclusion 5: The browser is 9x more important than the JavaScript library chosen.

I feel this is the most important conclusion here. You can debate about which JavaScript library is the fastest in certain situations, but ultimately, it doesn't even matter!! The browser itself is so much more important a factor in speed and performance than the actual library itself. Looking back at the normalized results from Figures 3 and 4, you can see that on average, the slowest of the three 'fast' libraries (Dojo) is only 15% slower than the fastest of the three (jQuery). Only 15% slower! However, if you look at the results for jQuery on the fastest browser (Chrome 1.0) and the slowest browser (IE 6), you can see that there's a 1000% difference!!! From these two numbers, the 15% difference seems like nothing compared to the

1000% difference. Hopefully, this should put an end to any debate between which of the three libraries are fastest, because ultimately, it doesn't really matter.

Conclusion 6: If JavaScript performance is important to your Web application, and you can control which browser your users use, do it.

In some situations, and some Web applications, you can control which browsers are used to visit sites. You might be lucky enough to be in that environment. I've worked on projects that were that lucky. In this situation, if you have a complex JavaScript application, or you think performance should be an issue, you should make an effort to control which browsers your users visit your Web application with. The results from these tests should be evident. If you have an intensive JavaScript application, and you can tell your users they *must* use Chrome, wouldn't you take that opportunity?

Conclusion 7: If you can't control which browsers your users use, keep performance on IE6 in mind.

Conversely, most of us live in a world where we can't control what browsers our users visit our site with. The sad truth is that there is a large amount of users who are still using IE 6 to do their surfing. As my tests show, the JavaScript engine in this browser is the slowest, by far, of the browsers I tested. However, because so many people are still using it, and good Web design calls for a "design to the lowest common denominator," it means you should design all of your JavaScript applications with IE 6 in mind.

jQuery performance tuning

The second part of this article looks into how to improve the performance of your jQuery code. The previous section showed that by choosing jQuery as your JavaScript library, you are making a good choice in terms of performance. Chances are that if you're reading this article, you've probably tried jQuery as well. But, just because the underlying library is fast doesn't mean that all the code you write will be of high quality. There's definitely an opportunity to write some really slow code using jQuery if you don't take a step back and think about what you're trying to do.

This section steps through some of the performance tuning and best practice tips for improving the speed of your jQuery code.

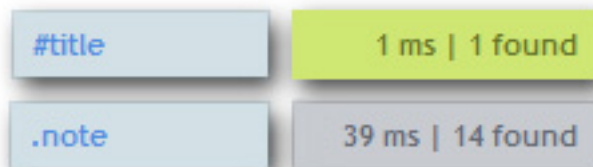
Tip #1 - Search by ID instead of by CLASS as often as possible

Two common search techniques used in jQuery code are to search through elements by their ID and to search through elements by their CLASS. Prior to JavaScript libraries, using regular JavaScript, it was still quite easy to find a page element by its ID. The method `getElementById()` could be used to find elements quickly. Finding CLASSES without a JavaScript library was tougher, though, and people tended to code around it by encoding things in their ID, if necessary. When using jQuery, searching for a CLASS is as easy as searching for an ID on a page,

thus making it seem that these two searches are interchangeable. However, this is not the case at all. Searching by ID is a lot faster than searching by CLASS. Turns out that jQuery simply uses the built-in `getElementById()` method when you search by ID, but has to iterate through every page element when searching by CLASS, looking for a match. Obviously, this indicates that larger and more complex pages will show a very slow response to searches by CLASS, while a search by ID shouldn't degrade as the page size increases.

This data is supported in the results for jQuery in the performance test I've been running. Let's look at the individual tests results to see where jQuery code needs attention. In this case, take a look at the test results when searching by ID and when searching by CLASS (Figure 5).

Figure 5. Search by ID versus CLASS



These tests aren't identical, but they do show numerical evidence that searching by ID is extremely fast compared to searching by CLASS. How does this affect your jQuery code? You should try to keep these things in mind when writing your searches: if you have the choice between an ID or a CLASS, always go with the ID. It also indicates that you should give all your elements an ID if they might be used in a search at any point in your code.

Listing 1 shows an actual jQuery test you can run to verify this on your own machine:

Listing 1. Class versus ID

```
$(document).ready(function() {  
  
    console.info("Start Test");  
    var d = new Date();  
    console.info(d.getSeconds() + " " + d.getMilliseconds());  
  
    var testBody = "";  
    for (var i=0; i<1000; i++)  
    {  
        testBody += "<div class='testable'+i+'>";  
    }  
    $("body").append(testBody);  
    for (var i=0; i<1000; i++)  
    {  
        $(".testable"+i);  
    }  
  
    var d = new Date();  
    console.info(d.getSeconds() + " " + d.getMilliseconds());  
    console.time("Start ID Test");
```

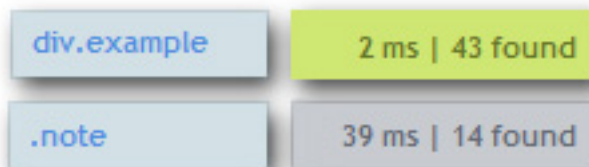
```
testBody = "";
for (var i=0; i<1000; i++)
{
  testBody += "<div id='testable'+i+'>";
}
$("body").append(testBody);
for (var i=0; i<1000; i++)
{
  $("#testable"+i);
}
var d = new Date();
console.info(d.getSeconds() + " " + d.getMilliseconds());
console.info("End Test");
});
```

The ID test took only 218 ms, while the CLASS test took 19.9 seconds!! Even on the simple page that's constructed for the test, the ID search is about 100x faster!

Tip #2 - Give as much search information as possible

jQuery gives you so many possible ways to search for page elements that sometimes it can be staggering to figure out which one is the best way to find your desired page element. One rule that will never be wrong is to provide as much information as you have to the search parameters. Thus, if you are searching for all the page elements with a CLASS of "clickable," you can increase the search performance if you know ahead of time that only DIVs will have that CLASS attached to them. Therefore, a search of "div.clickable" will be much faster. Figure 6 shows the numerical results to back this.

Figure 6. Supply as much information as possible



Given the underlying JavaScript code, this shouldn't be surprising. By supplying the element tag, the number of elements being searched for a match on the CLASS argument is drastically reduced, increasing performance to make it nearly as fast as a search by ID in this example.

Developers should not be lazy when writing their jQuery selections, although it is tempting to do because of its simplicity. The simplicity makes you lower your guard. By making the search mechanism so easy, it's tempting to simply enter one piece of information. However, you should always try to use as much information as possible, especially if the information is known. Listing 2 gives a good example.

Listing 2. Supplying enough information

```

// Assume there are 50 of these in some giant form, and you need to validate
// these fields before they are submitted, and there are hundreds of other
// elements on the page as well
<input type=text class="notBlank">

// the "bad" way to validate these fields
$(".notBlank").each(function(){
    if ($(this).val()=="")
        $(this).addClass("error");
});

// the "good" way to validate these fields
$("input.notBlank").each(function(){
    if ($(this).val()=="")
        $(this).addClass("error");
});

// the "best" way to validate these fields
$("input:text.notBlank").each(function(){
    if ($(this).val()=="")
        $(this).addClass("error");
});

```

Tip #3 - Cache your selectors

The final performance tip takes advantage of the fact that nearly all of the jQuery selectors return the jQuery object. This means that under ideal situations, you might only need to run the selection once, with the ability to daisy-chain your functions together, or cache the results for future use. You shouldn't feel constrained about caching either, because the actual objects themselves are small compared to the overall amount of memory available for storage.

Listing 3 shows a few examples of how to utilize caching.

Listing 3. Caching

```

// Imagine a function that hides all the div's with a class of "hideable"
// when a button is pressed. These DIV's might reappear later when
// working with the page, so the button can be pressed any number of
// times, and the DIV's that have reappeared
// will again be made to be hidden.

$("#ourHideButton").click(function(){
    $(".hideable").hide();
});

// As you saw in the CLASS versus ID example, though, a search for
// CLASS is very inefficient
// If this button is pressed often, it could lead to a slow response
// Instead of the above example, you should write your code like this

var hideable;

$("#ourHideButton").click(function(){
    if (hideable)
        hideable.hide();
    else
        hideable = $(".hideable").hide();
});

// You can cache your search in a JavaScript variable and reuse it every time
// the button is pressed. Because jQuery almost always returns the

```

```
// jQuery object, you can save it the first time it is called for future use
```

For my final example code on performance, take a look at the widget I talked about in the very first article in the series (see [Resources](#)). This widget was the checkbox in the top left corner of tables that allowed you to select/deselect all the checkboxes in that column. This is quite common on e-mail applications, allowing you to select/deselect all the messages.

Listing 4. Performance improvement

```
// Here is the code as I originally presented it in that article. Let's see
// if we can improve the performance in any way from the things we learned here

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

// Here's the improved function. The search for the ID of "selectall" is
// OK as-is, because we saw how fast the ID search is.
// The search for the CLASS of "selectable" was not well-designed though,
// because we saw a search by CLASS is very inefficient.
// First step was improving the search by supplying as much information as we know.
// We narrowed the search to only checkboxes with the CLASS of selectable.
// This should improve our search
// Further, we can cache this search because we will only need to perform it once
// Finally, we can perform this search before the selectall checkbox is even
// checked (when the page is finished loading), so that the search is completed
// and cached before the user even uses it.
// These 3 simple performance steps gave me a 200% increase in speed when tested
// on a page with 200 rows of data.

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

Final points on performance

The issue of speed and performance are by no means minor issues when it comes to working with JavaScript. And as the evidence shows, the developers who created jQuery and the ones working on the browser's built-in JavaScript engines are not blind to this fact either. In fact, I'd say that over the past six months, the single-most important issue in browser development has been JavaScript engine speed. All

signs are pointing to a very rapid increase in performance in JavaScript execution in the next year, with both the jQuery code and the JavaScript engine seeing drastic increases in speed. In fact, the news from these camps is that this is going to be the case exactly. John Resig, the guy who is leading the jQuery project, has talked about his new "Sizzle" selection engine. He has written a selection engine from scratch, and claims that preliminary results show it has a 4x increase in speed on Firefox. That's a very notable increase in speed! As I was writing this last section of the article, in fact, the jQuery 1.3 version was released and included the Sizzle selection engine. jQuery claims the 1.3 version of the selection engine has a 49% increase in speed over 1.2.6 when all the browsers are considered. Other claims of the 1.3 release are a 6x improvement in HTML injection (adding new elements into the DOM), and a 3x improvement in positioning functions. Exciting to have the major update to jQuery released on the day I'm finishing this article!

The other aspect of JavaScript performance is the browser, which I pointed out, is 9x more important than which library you choose. Firefox and Chrome have been battling back and forth about which JavaScript engine will be the fastest, with Safari getting into the mix as well. From my tests above, you saw that Chrome has a significant advantage over Firefox at this point, but the new Tracemonkey JavaScript engine that's going to be included in Firefox 3.1 is going to be (and this is their claim, not mine, and not made up) 20-40 times faster than the current JavaScript engine in 3.0. Wow, 20-40 times faster!!

Over the next year or two, you will see so many improvements in the underlying speed of the JavaScript engines and the libraries themselves that the speed of JavaScript-led Web applications will make even the most complex applications reasonable (except on IE6 of course).

Another thing to consider when you're deciding if you want to use a JavaScript library or hand-code the JavaScript yourself, is all the work and debugging that has gone into the JavaScript libraries. They've had hundreds of users for several years looking at every function in the library. You have yourself looking at that function you wrote at 2 a.m. after drinking lots of caffeinated drinks. Who do you trust more? Plus, even if you think that somehow you can write code that's faster than the jQuery library, do you think whatever imaginary performance hit you would have wouldn't be outweighed by all the advantages of using the library? You'd be willing to throw away all the huge conveniences of using jQuery and its library of functions because you want the added efforts and headaches of writing your own code, which will take you more time to write, produce more bugs, and all so you can feel "extra nerdy?" I'll pass, thank you.

Finally, my last point might depress some people, but you have to consider the people who are writing these jQuery libraries. They are some of the smartest and best programmers in the world, and every piece of super-smart code that they can write, and that you can't write, is going into these libraries. I am not too proud to admit that the people writing the jQuery library are a lot better programmers than

me. To think that I could write better and faster code than them is silly. Why not take advantage of all their smarts for yourself?

Conclusion

This article discussed the performance of jQuery, and JavaScript libraries in general. Through an extensive testing of the selection methods, you saw that even within the libraries themselves, there were great differences in the speeds with which they operated. You saw that jQuery can be considered one of the faster libraries you can choose. However, you also saw that even if you choose one of the fast JavaScript libraries, your Web application's performance is 9x more dependent on the browser your visitors use to run your Web application. If you have the opportunity to force your users to use a certain Web browser, do it. Find the browsers that run your Web applications fastest and push your users to take advantage of the inherent advantages they will see by doing this. Ideally, you can push out the slowest JavaScript browsers from existence, and really bring the next frontier of Web apps into existence.

I also gave you three performance tips for working with jQuery. Granted, there are a few sites out there that offer more tips, but these three tips are by far the most "bang for the buck" when it comes to performance gains. And three things are easier to remember than 50. The other tips are good as well, and I'll point out a few of them in [Resources](#), but if you give your code an audit with these three things in mind you will see a tremendous gain in your performance. The three jQuery tips you should always remember: a search by ID is 100x faster than a search by a CLASS, always supply as much information to a search as possible, and cache your selections whenever possible.

Finally, I gave a quick look at the upcoming changes you'll be seeing in jQuery and the browser's JavaScript engines. You saw that, as I was writing the end of this article, jQuery 1.3 was released, which promised tremendous performance improvements in selection and other aspects of the code. You also saw that Firefox promised a next-generation JavaScript engine that will be 20-40 times faster! All signs point to an extremely fast JavaScript environment in the next year or two. As a result, you should see extremely large and complex Web applications become more common, as the ability to run them well will become a reality.

In closing, the comment that sparked this article — "jQuery is nice on a simple page, but on a complex page, the performance is terrible. Until the performance issues are fixed, you have to use regular JavaScript for a complex page." — should be re-examined after all the evidence presented. I think based on what I've seen, the jQuery's "performance issues" would also be present in "regular JavaScript" itself. In fact, the real performance problems don't lie with jQuery or JavaScript, but with the browsers themselves. So I would agree that a complex page, with poorly designed jQuery code, on IE6, would have terrible performance. But, I'm hoping that I have

shown you that good jQuery, with a minimal number of performance tips, can run a very complex page on good browsers without any noticeable performance issues.

Resources

Learn

- Read the complete [jQuery API page](#) to see all of the available functions in the library.
- Read the [Release Notes for 1.3](#) to see all the changes and improvements in the latest release.
- Run [The SlickSpeed Test](#) on your own to verify the results for yourselves.
- Read [25 jQuery performance tips](#), which has many other tips on performance besides the big three presented in this article.
- Get a thorough and complete background on CSS, JavaScript, and any other Web language at [W3Schools](#).
- Read the first 3 articles in my jQuery series, that deal with an introduction to the library:
 - "[Working with jQuery, Part 1: Bringing desktop applications to the browser](#)" (developerWorks, September 2008)
 - "[Working with jQuery, Part 2: Building tomorrow's Web applications today](#)" (developerWorks, September 2008)
 - "[Working with jQuery, Part 3: Rich Internet applications with jQuery and Ajax : JQuery: Building tomorrow's Web apps today](#)" (developerWorks, October 2008)

Get products and technologies

- Download the [1.3.2 Minimized jQuery](#), which was the latest stable version at the time of this writing, and drop it in your own code.

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.