

Bringing your Web pages to Life with jQuery

How to use JavaScript in your web pages and stay sane

Tobias Oetiker

Digicomp OpenTuesday January 5th, 2010

1 Introduction

The Browser - Cross Platform in the Real World

- Applications in the Browser: Netscapes original Plan.
- JavaScript graduated with Web 2.0
- Speed: Nitro (Safari), V8 (Chrome), Tracemonkey (Firefox)
- IE: What is JavaScript ?

It is said that people at Netscape back in the nineties had the vision of escaping the Microsoft dominance by enhancing their browser so that it could become a platform of its own for running client side applications. It is also said that MS was not thrilled by this thought.

Netscape is no more but the vision has become a reality. The Web 2.0 hype sparked a slew of highly interactive web applications that used JavaScript snippets on the browser to enhance the user experience.

W3C Standards

- W3C standardizes HTML
- W3C standardizes DOM
- IE remains borderline compliant
- Differences in key event handling

jQuery features

- css like object selection
- abstracting away browser differences
- small code size
- good performance
- object manipulation

- event handling
- optical effects
- extensible with plugins
- MIT and GPL Licensed

2 Quick Start

hello world

Mixed html and javascript in php style

```
1 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
2 <head>
3 <script type="text/javascript" src="jquery-1.3.2.min.js">
4 </script>
5 </head>
6 <body>
7 <input
8   type="button"
9   onclick="jQuery('p').append(' and Good Bye') "
10  value="Click Me!"
11 />
12 <p>Hello</p><p>Another Paragraph</p>
13 </body>
14 </html>
```

hello-world.html

hello world cleaner

JavaScript in the header

```
1 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
2 <head>
3 <script type="text/javascript" src="jquery-1.3.2.min.js">
4 </script>
5 <script type="text/javascript">
6 function add(){
7   jQuery('p').append(' and Good Bye');
8 }
9 </script>
10 </head>
11 <body>
12 <input type="button" onclick="add()" value="Click Me!" />
13 <p>Hello</p><p>Another Paragraph</p>
14 </body>
15 </html>
```

hello-world2.html

hello world even cleaner

Only html in the html file

```
1 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
2 <head>
3 <script type="text/javascript" src="jquery-1.3.2.min.js">
4 </script>
5 <script type="text/javascript" src="hello-world3.js">
6 </script>
7 </head>
8 <body>
9 <input type="button" onclick="add()" value="Click Me!" />
10 <p>Hello</p><p>Another Paragraph</p>
11 </body>
12 </html>
```

hello-world3.html JavaScript in a separate file

```
1 // a little javascript function
2 function add(){
3     var t = ' Good Bye';
4     jQuery('p').append(t);
5 }
```

hello-world3.js

hello world in color

and then I got carried away ...

```
1 function colorGen(){
2     var col = '#';
3     for (var i=0;i<6;i++){
4         col += (Math.round(Math.random()*12))
5             .toString(16);
6     }
7     return col;
8 };
9
10 function add(){
11     var t = jQuery('<span/>')
12         .text(' Color')
13         .css('color', colorGen());
14     jQuery('p').append(t);
15 };
```

hello-world4.js

3 Order to the Chaos

Content and Design - HTML and CSS

- HTML structured content
- CSS layout control

JavaScript

- very simple
- a bit object oriented
- a life in the sandbox

DOM, BOM, Events

BOM Browser Object Model. Access the browser from JavaScript.

DOM Document Object Mode. Access the document from JavaScript.

Events call function when something happens.

jQuery

- DOM is standardized <http://www.w3.org/DOM/>
- to Vendors merely a suggestion
- cross Browser compatibility sketchy
- jQuery to the rescue !

4 The Document Object Model

The Document Object Model

- document access for javascript
- manipulation affects screen
- a W3C standard
- DOM1 1998, DOM2 2000, DOM3 2004

DOM Features that do work

- HTML/XML access/manipulation
- CSS access
- Events (UI, Mouse, Keyboard (somewhat), HTML)

DOM Features that do not generally work

- Loading of additional documents
- Structural Validation
- see <http://www.w3.org/2003/02/06-dom-support.html>

DOM Scripting Example

```
1 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
2 <head>
3 <script type="text/javascript">
4 function ds(){
5     var d=document;
6     d.getElementById('myId').innerHTML="Hello";
7     var a = d.getElementsByTagName('div');
8     for (var i=0;i<a.length;i++){
9         a[i].innerHTML+="&lt;"+a[i].innerHTML+"&gt;";
10    }
11 }
12 </script>
13 </head>
14 <body>
15 <input type="button" onclick="ds()" value="Click Me!" />
16 <div>tag 1</div><div>tag 2</div>
17 <p id="myId">myId P tag</p>
18 </body>
19 </html>
```

dom-script.html

5 jQuery Basics

What else do you need ?

- Structure for JavaScript programs
- Easy DOM element pickup
- Browser Abstraction

the jQuery function

There are three ways to call jQuery

```
1 obj=jQuery(selector);    // Select Nodes
2 obj=jQuery('<tag>');      // Create Node
3 obj=jQuery(dom_element); // Wrap Node
```

all return a jQuery object

jQuery Dom Selectors

Select matching elements from the document

```
1 var x = jQuery(selector);
```

selector syntax

```
1 #xx      <div id="xx"/>
2 div      <div/>
3 .yy      <div class="yy"/>
4 *        matches everything
```

selector selects everything that matches

The jQuery Object

- acts on all matching DOM objects/elements
- access raw DOM objects/elements
- manipulate matching elements
- apply css
- run optical effects
- manage event callbacks
- Ajax calls

DOM Scripting Example in jQuery

```
1 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
2 <head>
3 <script type="text/javascript" src="jquery-1.3.2.min.js">
4 </script>
5 <script type="text/javascript">
6 function ds() {
7     jQuery('#myId').html('Hello');
8     jQuery('div').each(function() {
9         var j = jQuery(this);
10        j.text('<'+j.text()+>');
11    });
12 }
13 </script>
14 </head>
15 <body>
16 <input type="button" onclick="ds()" value="Click Me!" />
```

```

17 <div>tag 1</div><div>tag 2</div>
18 <p id="myId">myId P tag</p>
19 </body>
20 </html>

```

dom-script-jquery.html

What about the Dollar

```

1 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
2 <head>
3 <script type="text/javascript" src="jquery-1.3.2.min.js">
4 </script>
5 <script type="text/javascript">
6 function ds() {
7     $('#myId').html('Hello');
8     $('div').each(function() {
9         var j = $(this);
10        j.text('<'+j.text()+>');
11    });
12 }
13 </script>
14 </head>
15 <body>
16 <input type="button" onclick="ds()" value="Click Me!" />
17 <div>tag 1</div><div>tag 2</div>
18 <p id="myId">myId P tag</p>
19 </body>
20 </html>

```

dom-script-jquery-dollar.html

var \$ = jQuery

- jQuery(x) == \$(x)
- invented to confuse perl and php programmers
- \$ is just another character
- potential for conflict as \$ is a popular choice

Use this instead ...

```

1 jQuery.noConflict();
2 (function ($) {
3     $('div') ....
4 })(jQuery);

```

Starting the script

- scripts get executed immediately
- body onload occurs when page is displayed
- use the `ready` method to run when DOM is ready (before the page is displayed)

```
1 jQuery.noConflict();
2 (function ($) {
3     function hello () {
4         $("p").text("a beautiful document");
5         window.alert("Document is ready");
6     };
7     $(document).ready(hello);
8 })(jQuery);
```

ready.html

6 JavaScript Fundamentals

A function pointer example

```
1 var add_alert=function(a,b){
2     window.alert('The Result is: '+a+b);
3 }
4 add_alert(1,2);
```

this is equivalent to

```
1 function add_alert(a,b){
2     ...
3 }
```

but it seems more logical.

Using the `function` keyword creates a function object which I can use exactly like a normal function.

Scoping for the naïve

You know scoping from other languages

```
1 var j = 100;
2 var z = 22;
3 for (var z = 1; z<10; z++){
4     var j = z;
5 }
6 alert('And the Winner is j:'+j+' z:'+z);
```

Or so you thought



Scoping for the disillusioned

JavaScript scoping works only within function blocks.

```
1 var j = 100;
2 var z = 22;
3 for (var z = 1; z < 10; z++) { (function () {
4     var j = z;
5 }) () }
6 alert ('And the Winner is j:' + j + ' z:' + z);
```

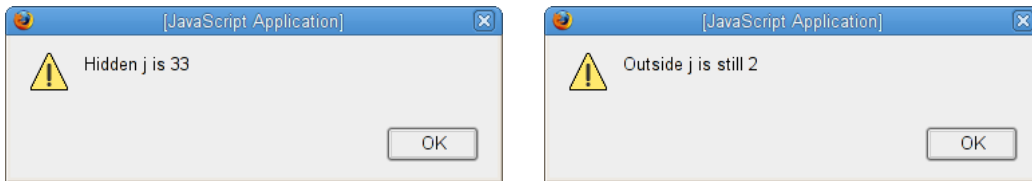


Note that z is outside the function block!

While many languages provide scope for every block, this is not the case for JavaScript. Here the only scoping block is the function block. So if we need scope inside a loop for example, we just add an anonymous function and execute it in place. I am not sure how efficient this is, so do not use it in tight loops.

A simple closures example

```
1 var set;
2 var get;
3 var j=2;
4 (function() { // the magic javascript scope trick
5     var j;
6     set = function(x) {j=x};
7     get = function() {alert ('Hidden j is '+j)};
8 }) (); // end of scope
9 set(33);
10 get();
11 alert ('Outside j is still '+j);
```



Everything as expected.

With closures, several functions can share a common variable defined outside the function. Lisp/Scheme has been the first language to implement this concept. Today many object oriented languages can do closures: There are subtle differences though, as I had to discover the hard way, working with JavaScript coming from a Perl/C background.

A broken function factory

```

1 var j = [];
2 for (var z = 1; z < 10; z++) {
3   var g = z;
4   j[g] = function(a) {
5     window.alert('The value for j[' + a + '] is ' + g);
6   };
7 }
8 j[2](2);

```



JavaScript! Has! No! Scope!

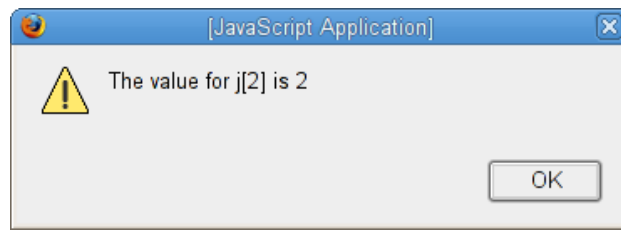
For! Loop! Blocks!

A working function factory

```

1 var j = [];
2 for (var z = 1; z < 10; z++) { (function() { // for a block
3   var g = z;
4   j[g] = function(a) {
5     window.alert('The value for j[' + a + '] is ' + g);
6   };
7 })() } // end of block
8 j[2](2);

```



Again the anonymous function trick comes to the rescue.

Scoping and closures are the most powerful concepts in JavaScript since they are an essential ingredient to making callback functions really useful. It took me almost a year to finally figure out about JavaScript scoping, before it was just the odd problem which I could not really fix and spent hours working finding a work-a-round, cursing mysterious JavaScript bugs in the process.

the call and apply methods

```
1 var x = 'document';
2 (function ($) {
3     function demo () {
4         function show_x (y, z) {
5             $ ("body")
6                 .append("<div> x=" +this.x+", "
7                     +"y="+y+", z="+z+"</div>");
8         }
9         var o = {
10            x: 'inner'
11        };
12        show_x ('a', 'b');
13        show_x.call (o, 'call_a', 'call_b');
14        show_x.apply (o, ['apply_a', 'apply_b']);
15    };
16    // calling demo when document is ready
17    $ (document).ready (demo);
18 }) (jQuery);
```

callapply.html

7 The magic this

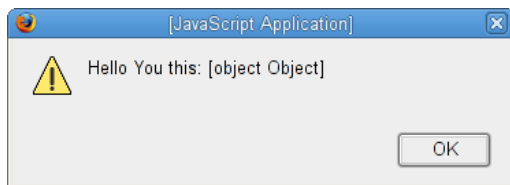
fun with this

```
1 var hello = {
2     world: 'You',
3     show: function () {
4         window.alert ('Hello '+this.world+' this: '+this); }
5 };
```

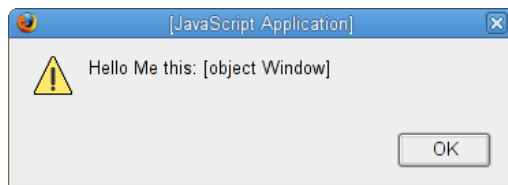
```

6 hello.show();           // method call
7 var plain = hello.show; // function pointer
8 var world = 'Me';      // change this.world
9 plain();               // method call

```



this refers to
the hello object.



this refers to the browsers
base Window object.

Be careful when using `this` in an anonymous function since it always points to the current parent. Call-back functions are especially 'vulnerable' to this problem.

8 JavaScript Coding Style

- Good Code looks Good
- is neither required
- nor sufficient
- but still

Writing beautiful Code

1. declare variables `var variable;`
2. avoid anonymous functions
3. do not use **with**
4. avoid name space pollution
5. limit line length to 80 chars
6. stick to `[_a-zA-Z0-9]` for identifiers

Test with <http://jshint.com/> — don't cry!

Naming conventions

- use CAPITALS for global variables/functions
- start variables and functions with a lowercase letter
- start new-able classes with a Capital letter
- names are only convention!

read <http://javascript.crockford.com/code.html>

9 jQuery Ajax

It is all about XMLHttpRequest

- making HTTP requests from javascript
- appeared in IE5 as XMLHttpRequest
- since 2005 a W3C standard DOM as XMLHttpRequest
- in IE7 Microsoft joined the party
- compatibility hell ...

Ajax without jQuery

```
1 if (typeof XMLHttpRequest === "undefined") {
2     function XMLHttpRequest() {
3         var obj_arr = [
4             "Msxml2.XMLHTTP.6.0",
5             "Msxml2.XMLHTTP.3.0",
6             "Msxml2.XMLHTTP",
7             "Microsoft.XMLHTTP"
8         ];
9         for (var i=0;i<4;i++){
10            try { return new ActiveXObject(obj_arr[i]); }
11            catch(e) { };
12        }
13        throw new Error("no support for XMLHttpRequest!");
14    };
15 }
```

jQuery abstracts this sort of code away.

jQuery load method

add the body of a document into the dom

```
1 jQuery.noConflict();
2 (function ($) {
3     function load_src() {
4         $("body").load("load.body.html");
5     }
6     $(document).ready(load_src);
7 })(jQuery);
```

the input file

```
1 <html>
2 <body>
3 Ajax Works!
4 </body>
5 </html>
```

load.html, load.body.html

Beware: Browser interprets this file!

jQuery ajax method

```
1 jQuery.noConflict();
2 (function ($) {
3     function handle_data(data, textStatus) {
4         $("#source").text(data);
5         $("#status").text("Status: "+textStatus);
6     };
7     $(document).ready(function () {
8         $.ajax({
9             url: "ajax.html",
10            dataType: "text",
11            cache: false,
12            success: handle_data
13        });
14    });
15 })(jQuery);
```

ajax.html

See also: \$.get, \$.post, \$.getJSON, \$.getScripts, \$.post

10 Debugging

debugging JavaScript

- No anonymous functions!
- View source is no help anymore!
- Firebug for Firefox
- Safari Web Inspector
- Chrome Developer Tools
- Opera 10 JavaScript tools
- IE Development Toolbar

I find that explicitly writing out the content of some variables can be quicker than tracking the code with a debugger. JavaScript provides several methods for doing that.

create debug output

```

1 window.alert("test window.alert");
2 try {
3   window.console.log("This is a test log");
4   window.console.warn("Test warning");
5   window.console.error("Test error");
6 } catch(err) {
7   window.alert("UUPS: "+err);
8 }
9
10 try {
11   throw "Ball";
12 } catch(err) {
13   window.alert("caught "+err);
14 }

```

logger.html

11 jQuery Plugins

using jQuery plugins

```

1 <script type="text/javascript"
2   src="jquery-1.3.2.min.js"></script>
3 <script type="text/javascript"
4   src="jquery-ui-1.7.2.custom.min.js"></script>
5 <link type="text/css"
6   href="css/jquery-ui-1.7.2.custom.css"
7   rel="stylesheet" />

```

```

8 <script type="text/javascript">
9 jQuery(document).ready(function(){
10     jQuery('#datepicker').datepicker({
11         numberOfMonths: 1,
12         showButtonPanel: false
13     });
14 });

```

datepicker.html

See <http://plugins.jquery.com/>, <http://jqueryui.com>

the logger plugin (utility)

```

1 if (typeof window.console !== 'undefined'
2     && typeof window.console.log !== 'undefined'){
3     jQuery.extend({
4         log : function (){
5             // 'arguments' is no real array pointer
6             // calling the array slice method on it.
7             // since arguments is no Array I have take
8             // the method from the Array prototype.
9             var args = Array.prototype.slice.call(arguments);
10            console.log.apply(console, args);
11        }
12    });
13 } else {
14     // if there is no console, do nothing
15     jQuery.extend({
16         log : function (){}
17    });
18 }
19 $.log('test message');
20 $.log("%s %d", 'another test message', 22);

```

logger-plugin.html

the redify plugin (action)

```

1 (function($){
2     $.fn.redify = function() {
3         var redden = function(){
4             $(this).css('color', 'red').fadeIn('slow');
5         };
6         return this.each(function() {
7             $(this).fadeOut('slow', redden);
8         });
9     };
10    $(document).ready(function(){
11        $(' .red').redify();
12    });
13 })(jQuery);

```

a plugin development pattern

1. Limit namespace pollution
2. Distinguish between utility and action methods
3. Return the jQuery object whenever possible
4. Use named arguments for non default options
5. Provide public access to default options
6. Keep private functions private
7. Support the metadata plugin
8. Use CamelCase
9. When `el` is a DOM element use `$el` for the jQuery object
10. Start namespaces with a lower case letter

inspired by Mike Alsup

jquery.PluginPattern.js

```

1  (function($) {
2      // setup a namespace for us
3      var nsp = 'pluginPattern';
4
5      // Public Variables and Methods
6      $[nsp] = {
7          // let the user override the default
8          // $.pluginPattern.defaultOptions.optA = false
9          defaultOptions: {
10             optA : true
11         },
12         // $.pluginPattern.publicVariable
13         publicVariable: false,
14         // $.pluginPattern.utilityMethod(...)
15         utilityMethod: function() {
16             // Do Something Publicly
17             window.alert('called pullic Utility Method');
18         }
19     };
20

```

```

21 // Private Variables and Functions in the _ object
22 // note that this will refer to _ unless you
23 // call using the call or apply methods
24 var _ = {
25     // _.varA
26     varA: 'varA',
27     // _.funcA
28     funcA: function () {
29         alert('called private functionA');
30     },
31     // _.init
32     init : function(){
33         window.alert('initialized');
34     }
35 };
36
37
38
39
40
41
42
43 // $(x).pluginPatternActionA(...)
44 $.fn[nsp+'ActionA'] = function(argA,argB,opts) {
45     var lop = $.extend(
46         {}, // start with an empty map
47         $.fn[nsp].defaultOptions, // add defaults
48         opts // add options
49     );
50
51
52
53
54
55
56
57
58
59
60
61
62
63     function actionA(){
64         var meta_opts = lop;
65         // lets you override the options
66         // inside the dom objects class property
67         // requires the jQuery metadata plugin
68         // <div class="hello {color: 'red'}">d</div>
69         if ($.meta){
70             meta_opts = $.extend({}, lop, $this.data());
71         }
72         // $this to access the jQuery object
73         var $this = $(this);
74         // per dom node context data
75         if (! this[nsp]){

```

```
76         this[nsp] = {};
77     }
78     this[nsp].contextVariable = 'bla';
79     // do some work
80     window.alert('called Action A on:' + $this.text());
81 };
82 return this.each(actionA);
83 };
84
85
86
87
88     //Initialization Code when DOM is ready
89     $(document).ready(_.init);
90
91 }) (jQuery);
```

<http://plugins.jquery.com/project/PluginPattern>