jQuery Coding Standards & Best Practices

Loading jQuery

1. Always try to use a CDN to include jQuery on your page. CDN Benefits (http://www.sitepoint.com/7-reasons-to-use-a-cdn/)

```
<script type="text/javascript" src="//ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js"></script>
<script>window.jQuery || document.write('<script src="js/jquery-2.1.1.min.js" type="text/javascript"><\/script>')</script>
```

   Click here for a list of popular jQuery CDNs.
2. Implement a fallback to your locally hosted library of same version as shown above. More Info (http://css-tricks.com/snippets/jquery/fallback-for-cdn-hosted-jquery/)
3. Use protocol-relative/protocol-independent (http://www.paulirish.com/2010/the-protocol-relative-url/) URL (leave `http:` or `https:` out) as shown above.
4. If possible, keep all your JavaScript and jQuery includes at the bottom of your page. More Info (http://developer.yahoo.com/blogs/ydn/high-performance-sites-rule-6-move-scripts-bottom-7200.html) and a sample on HTML5 Boilerplate (https://github.com/h5bp/html5-boilerplate/blob/master/src/index.html).
5. What version to use?
   - DO NOT use jQuery version 2.x if you support Internet Explorer 6/7/8.
   - For new web-apps, if you do not have any plugin compatibility issue, it's highly recommended to use the latest jQuery version.
   - When loading jQuery from CDN's, always specify the complete version number you want to load (Example: *1.11.0* as opposed to *1.11* or just *1*).
   - DO NOT load multiple jQuery versions.
   - DO NOT use jquery-latest.js from jQuery CDN (http://blog.jquery.com/2014/07/03/dont-use-jquery-latest-js/).
6. If you are using other libraries like Prototype, MooTools, Zepto etc. that uses `$` sign as well, try not to use `$` for calling jQuery functions and instead use `jQuery` simply. You can return control of `$` back to the other library with a call to `$.noConflict()`.
7. For advanced browser feature detection, use Modernizr (http://modernizr.com/).

jQuery Variables

1. All variables that are used to store/cache jQuery objects should have a name prefixed with a `$`.
2. Always cache your jQuery selector returned objects in variables for reuse.

```
var $myDiv = $("#myDiv");
$myDiv.click(function(){...});
```

3. Use camel case (//en.wikipedia.org/wiki/CamelCase) for naming variables.

Selectors

1. Use ID selector whenever possible. It is faster because they are handled using `document.getElementById()` .
2. When using class selectors, don't use the element type in your selector. Performance Comparison (http://jsperf.com/jqeury-selector-test)

```
var $products = $("div.products"); // SLOW
var $products = $(".products"); // FAST
```

3. Use find for *Id->Child* nested selectors. The `.find()` approach is faster because the first selection is handled without going through the Sizzle selector engine. More Info (http://learn.jquery.com/performance/optimize-selectors/)

```
// BAD, a nested query for Sizzle selector engine
var $productIds = $("#products div.id");

// GOOD, #products is already selected by document.getElementById() so only div.id n
eeds to go through Sizzle selector engine
var $productIds = $("#products").find("div.id");
```

4. Be specific on the right-hand side of your selector, and less specific on the left. More Info (http://learn.jquery.com/performance/optimize-selectors/)

```
// Unoptimized
$("div.data .gonzalez");

// Optimized
$(".data td.gonzalez");
```

5. Avoid Excessive Specificity. More Info (http://learn.jquery.com/performance/optimize-selectors/), Performance Comparison (http://jsperf.com/avoid-excessive-specificity)

```
$(".data table.attendees td.gonzalez");

// Better: Drop the middle if possible.
$(".data td.gonzalez");
```

6. Give your Selectors a Context.

```
// SLOWER because it has to traverse the whole DOM for .class
$('.class');

// FASTER because now it only looks under class-container.
$('.class', '#class-container');
```

7. Avoid Universal Selectors. More Info □ (http://learn.jquery.com/performance/optimize-selectors/)

```
$('div.container > *'); // BAD
$('div.container').children(); // BETTER
```

8. Avoid Implied Universal Selectors. When you leave off the selector, the universal selector (*) is still implied. More Info □ (http://learn.jquery.com/performance/optimize-selectors/)

```
$('div.someclass :radio'); // BAD
$('div.someclass input:radio'); // GOOD
```

9. Don’t Descend Multiple IDs or nest when selecting an ID. ID-only selections are handled using `document.getElementById()` so don't mix them with other selectors.

```
$('#outer #inner'); // BAD
$('div#inner'); // BAD
$('.outer-container #inner'); // BAD
$('#inner'); // GOOD, only calls document.getElementById()
```

DOM Manipulation

1. Always detach any existing element before manipulation and attach it back after manipulating it. More Info □ (http://learn.jquery.com/performance/detach-elements-before-work-with-them/)

```
var $myList = $("#list-container > ul").detach();
//...a lot of complicated things on $myList
$myList.appendTo("#list-container");
```

2. Use string concatenation or `array.join()` over `.append()`. More Info □ (http://learn.jquery.com/performance/append-outside-loop/)
Performance comparison: http://jsperf.com/jquery-append-vs-string-concat □ (http://jsperf.com/jquery-append-vs-string-concat)

```
// BAD
var $myList = $("#list");
for(var i = 0; i < 10000; i++){
    $myList.append("<li>"+i+"</li>");
}

// GOOD
var $myList = $("#list");
var list = "";
for(var i = 0; i < 10000; i++){
    list += "<li>"+i+"</li>";
}
$myList.html(list);

// EVEN FASTER
var array = [];
for(var i = 0; i < 10000; i++){
    array[i] = "<li>"+i+"</li>";
}
$myList.html(array.join(''));
```

3. Don't Act on Absent Elements. More Info (http://learn.jquery.com/performance/dont-act-on-absent-elements/)

```
// BAD: This runs three functions before it realizes there's nothing in the selectio
n
$("#nosuchthing").slideUp();

// GOOD
var $mySelection = $("#nosuchthing");
if ($mySelection.length) {
    $mySelection.slideUp();
}
```

Events

1. Use only one Document Ready handler per page. It makes it easier to debug and keep track of the behavior flow.
2. DO NOT use anonymous functions to attach events. Anonymous functions are difficult to debug, maintain, test, or reuse. More Info (http://learn.jquery.com/code-organization/beware-anonymous-functions/)

```
$("#myLink").on("click", function(){...}); // BAD

// GOOD
function myLinkClickHandler(){...}
$("#myLink").on("click", myLinkClickHandler);
```

3. Document ready event handler should not be an anonymous function. Once again, anonymous functions are difficult to debug, maintain, test, or reuse.

```
$(function(){ ... }); // BAD: You can never reuse or write a test for this function.

// GOOD
$(initPage); // or $(document).ready(initPage);
function initPage(){
    // Page load event where you can initialize values and call other initializers.
}
```

4. Document ready event handlers should be included from external files and inline JavaScript can be used to call the ready handle after any initial setup.

```
<script src="my-document-ready.js"></script>
<script>
        // Any global variable set-up that might be needed.
        $(document).ready(initPage); // or $(initPage);
</script>
```

5. DO NOT use behavioral markup in HTML (JavaScript inlining), these are debugging nightmares. Always bind events with jQuery to be consistent so it's easier to attach and remove events dynamically.

```
<a id="myLink" href="#" onclick="myEventHandler();">my link</a> <!-- BAD -->
```

```
$("#myLink").on("click", myEventHandler); // GOOD
```

6. When possible, use custom namespace (http://api.jquery.com/event.namespace/) for events. It's easier to unbind the exact event that you attached without affecting other events bound to the DOM element.

```
$("#myLink").on("click.mySpecialClick", myEventHandler); // GOOD
// Later on, it's easier to unbind just your click event
$("#myLink").unbind("click.mySpecialClick");
```

7. Use event delegation (http://learn.jquery.com/events/event-delegation/) when you have to attach same event to multiple elements. Event delegation allows us to attach a single event listener, to a parent element, that will fire for all descendants matching a selector, whether those descendants exist now or are added in the future.

```
$("#list a").on("click", myClickHandler); // BAD, you are attaching an event to all
the links under the list.
$("#list").on("click", "a", myClickHandler); // GOOD, only one event handler is atta
ched to the parent.
```

Ajax

1. Avoid using *.getJson()* or .get(), simply use the $.ajax() as that's what gets called internally.

2. DO NOT use *http* requests on *https* sites. Prefer schemaless URLs (leave the protocol *http/https* out of your URL)
3. DO NOT put request parameters in the URL, send them using data object setting.

```
// Less readable...
$.ajax({
    url: "something.php?param1=test1&param2=test2",
    ....
});

// More readable...
$.ajax({
    url: "something.php",
    data: { param1: test1, param2: test2 }
});
```

4. Try to specify the *dataType* setting so it's easier to know what kind of data you are working with. (See Ajax Template example below)
5. Use Delegated event handlers for attaching events to content loaded using Ajax. Delegated events have the advantage that they can process events from descendant elements that are added to the document at a later time (example Ajax). More Info □ (http://api.jquery.com/on/#direct-and-delegated-events)

```
$("#parent-container").on("click", "a", delegatedClickHandlerForAjax);
```

6. Use Promise interface: More Examples □ (http://www.htmlgoodies.com/beyond/javascript/making-promises-with-jquery-deferred.html)

```
$.ajax({ ... }).then(successHandler, failureHandler);

// OR
var jqxhr = $.ajax({ ... });
jqxhr.done(successHandler);
jqxhr.fail(failureHandler);
```

7. Sample Ajax Template: More Info □ (https://api.jquery.com/jQuery.ajax/)

```
var jqxhr = $.ajax({
    url: url,
    type: "GET", // default is GET but you can use other verbs based on your needs.
    cache: true, // default is true, but false for dataType 'script' and 'jsonp', so
 set it on need basis.
    data: {}, // add your request parameters in the data object.
    dataType: "json", // specify the dataType for future reference
    jsonp: "callback", // only specify this to match the name of callback parameter
your API is expecting for JSONP requests.
    statusCode: { // if you want to handle specific error codes, use the status code
 mapping settings.
        404: handler404,
        500: handler500
    }
});
jqxhr.done(successHandler);
jqxhr.fail(failureHandler);
```

Effects and Animations

1. Adopt a restrained and consistent approach to implementing animation functionality.
2. DO NOT over-do the animation effects until driven by the UX requirements.
    - Try to use simeple show/hide, toggle and slideUp/slideDown functionality to toggle elements.
    - Try to use predefined animations durations of "slow", "fast" or 400 (for medium).

Plugins

1. Always choose a plugin with good support, documentation, testing and community support.
2. Check the compatibility of plugin with the version of jQuery that you are using.
3. Any common reusable component should be implemented as a jQuery plugin. Click here for jQuery Plugin Boilerplate code.

Chaining

1. Use chaining as an alternative to variable caching and multiple selector calls.

    ```
    $("#myDiv").addClass("error").show();
    ```

2. Whenever the chain grows over 3 links or gets complicated because of event assignment, use appropriate line breaks and indentation to make the code readable.

```
$("#myLink")
    .addClass("bold")
    .on("click", myClickHandler)
    .on("mouseover", myMouseOverHandler)
    .show();
```

3. For long chains it is acceptable to cache intermediate objects in a variable.

Miscellaneous

1. Use Object literals for parameters.

```
$myLink.attr("href", "#").attr("title", "my link").attr("rel", "external"); // BAD,
3 calls to attr()
// GOOD, only 1 call to attr()
$myLink.attr({
    href: "#",
    title: "my link",
    rel: "external"
});
```

2. Do not mix CSS with jQuery.

```
$("#mydiv").css({'color':red, 'font-weight':'bold'}); // BAD
```

```
.error { color: red; font-weight: bold; } /* GOOD */
```

```
$("#mydiv").addClass("error"); // GOOD
```

3. DO NOT use Deprecated Methods. It is always important to keep an eye on deprecated methods for each new version and try avoid using them. Click here (http://api.jquery.com/category/deprecated/) for a list of deprecated methods.
4. Combine jQuery with native JavaScript when needed. See the performance difference for the example given below: http://jsperf.com/document-getelementbyid-vs-jquery/3 (http://jsperf.com/document-getelementbyid-vs-jquery/3)

```
$("#myId"); // is still little slower than...
document.getElementById("myId");
```