5 Things You Should Stop Doing With jQuery

by **Burke Holland** on May 6, 2013

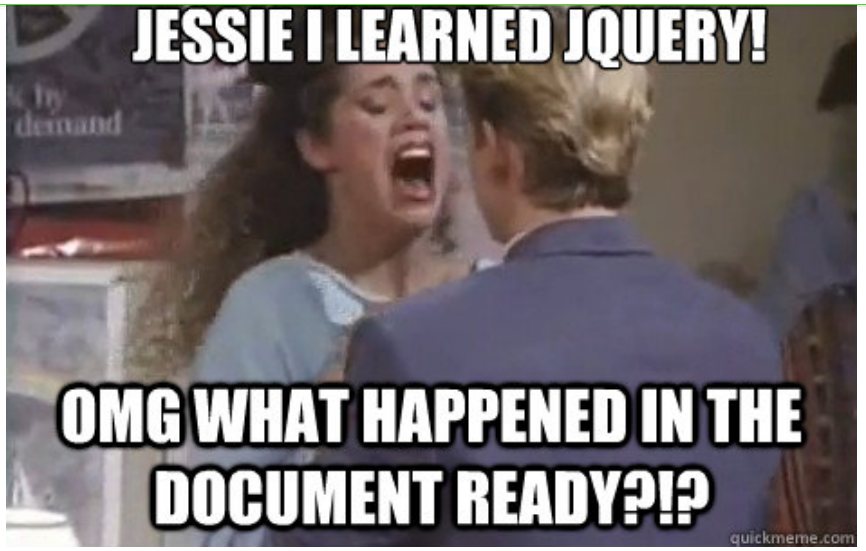
Frontend Development 93 Comments

The modern web is always changing, and this article is more than two years old.

By Burke Holland

When I first started using jQuery, I was so excited. I was using vanilla JS and really struggling with understanding when elements on a page were ready and how I could access them. When I learned about jQuery, I think I did what many people do. With tears of joy running down my face, I opened up a document ready function and then just vomited up massive amounts of jQuery.

Some of the worst jQuery code ever written was written by me – I can assure you of that.



Since then, I've spent a few years at the Bayside of jQuery. I've had my heart broken a few times and learned a some things along the way. In doing so, I'd like to share 5 things in jQuery that I think you should think twice about doing.

1. Stop Using Document Ready

Back in the day of heavy server frameworks, knowing that your page was fully constructed before you tried to mutate it was a big deal. This was especially true if you were using partial views of some sort that were injected into the layout/master page by the server at runtime.

Nowadays, it's considered best practice to include your scripts at the bottom of the page. The HTML5 async attribute notwithstanding, scripts are loaded and executed synchronously by the browser. That means that if you have a large script in the head of your page, it will delay loading of the DOM and make your page seem slower than it actually is. Loading all scripts last will at least make your application seem to load faster. It also gives you a chance to use a spinner or loading overlay if you are completely dependent on JavaScript for your UI.

If you are adhering to the "scripts at the bottom" best practice, then you have no need for jQuery's document ready function as the HTML is already loaded by the time the script is run.

```
<p id="zack">This element is on the page <strong>BEFORE</strong> a

  <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.9.1/

  <script type="text/javascript" charset="utf-8">

    // if you include your scripts at the very bottom, you don't n
    (function($) {

      $("#zack").css("color", "green");
      $("#slator").css("color", "red");

    }(jQuery));

  </script>
```

```
<p id="slater">This element comes after the scripts and won't be a
```

2. Stop Using The Wrong Iterator

I've got no hard evidence to back this up, but I'd be willing to guess that the each function is the most used of the jQuery utility methods. I base this on how often I am tempted to use it and how often I see it in other people's code. It's incredibly convenient for looping over DOM elements and JavaScript collections. It's also terse. There is nothing wrong with it, except that its not the only iteration function the jQuery provides. People use each like it's Zack Morris, at which point every looping problem starts looking like Kelly Kapowski.

MAP

If you have an array of items and you want to loop through it and filter out some of the items, you might be tempted to use the each method like so:

```
(function($) {

  var allStarCast = [
    { firstName: "Zack", lastName: "Morris" },
    { firstName: "Kelly", lastName: "Kapowski" },
    { firstName: "Lisa", lastName: "Turtle" },
    { firstName: "Screech", lastName: "Powers" },
    { firstName: "A.C.", lastName: "Slater" },
    { firstName: "Jessie", lastName: "Spano" },
    { firstName: "Richard", lastName: "Belding" }
  ]

  // iterate through the cast and find zack and kelly
  var worldsCutestCouple = [];
  $.each(allStarCast, function(idx, actor) {
    if (actor.firstName === "Zack" || actor.firstName === "Kelly")
      worldsCutestCouple.push(actor);
    }
  });

  console.log(worldsCutestCouple);

}(jQuery));
```

Try It

That works, but jQuery actually provides a method specifically for this scenario and it's called map. What it does is takes any items that get returned and adds them to a new array. The looping code then looks like this…

```
(function($) {

  var allStarCast = [
    { firstName: "Zack", lastName: "Morris" },
    { firstName: "Kelly", lastName: "Kapowski" },
    { firstName: "Lisa", lastName: "Turtle" },
    { firstName: "Screech", lastName: "Powers" },
    { firstName: "A.C.", lastName: "Slater" },
    { firstName: "Jessie", lastName: "Spano" },
    { firstName: "Richard", lastName: "Belding" }
  ]

  // iterate through the cast and find zack and kelly

  var worldsCutestCouple = $.map(allStarCast, function(actor, idx)
    if (actor.firstName === "Zack" || actor.firstName === "Kelly")
      return actor;
    }
  });

  console.log(worldsCutestCouple);

}(jQuery));
```

Try It

Did you notice that the arguments for the .map() callback and the .each() callback are reversed? Watch out for that because it can bite you when you start using map.

The map function allows you to modify the items before they get slapped into the new array. I'm not currently doing that, so technically I'm still not using the "right" iterator. I should really be using grep.

GREP

If you spend most of your days on Windows, then this term might be foreign to you. The term is typically used for a Unix command line utility for searching files that contain text matching a given regular expression.

In jQuery, grep is a function whose purpose is to reduce a collection by removing elements that don't pass the test in the callback. The provided callback returns a boolean value. Return true if you want the item to remain, and false(y) if you don't. It will not affect the original array. You cannot modify the items as you reduce them.

```
(function($) {

  var allStarCast = [
      { firstName: "Zack", lastName: "Morris" },
      { firstName: "Kelly", lastName: "Kapowski" },
      { firstName: "Lisa", lastName: "Turtle" },
```

```
      { firstName: "Screech", lastName: "Powers" },
      { firstName: "A.C.", lastName: "Slater" },
      { firstName: "Jessie", lastName: "Spano" },
      { firstName: "Richard", lastName: "Belding" }
  ]

  // iterate through the cast and find zack and kelly

  var worldsCutestCouple = $.grep(allStarCast, function(actor) {
      return (actor.firstName === "Zack" || actor.firstName === "K
  });

  console.log(worldsCutestCouple);

}(jQuery));
```

Try It

3. Stop Using "this"

This isn't a jQuery issue, but jQuery can definitely exacerbate the problem. The context of "this" is always changing. jQuery sometimes changes the context to something that you might not be expecting. In the each callback, this is the current item or element in the collection. In the map function, it's the window object. You can see how you could get yourself confused pretty quickly.

Have a look at the following example and then we'll talk about why it blows up.

```
(function($) {

  var sbtb = {
    log: function(message) {
      $("#log").append("").html(message);
    },
    cast: [
      { firstName: "Zack", lastName: "Morris", soExcited: false },
      { firstName: "Kelly", lastName: "Kapowski", soExcited: true
      { firstName: "Lisa", lastName: "Turtle", soExcited: true },
      { firstName: "Screech", lastName: "Powers", soExcited: false
      { firstName: "A.C.", lastName: "Slater", soExcited: false },
      { firstName: "Jessie", lastName: "Spano", soExcited: true },
      { firstName: "Richard", lastName: "Belding", soExcited: fals
    ],
    soExcited: function() {

    // use "this" to get a reference to the cast on this object
    $.each(this.cast, function(idx, actor) {
```

```
      // call the log function
      this.log(actor.firstName + " " + actor.lastName); // BOOM! S
      // "this" is now a cheesy actor, not the sbtb object anymore

    });

    }
  };

  sbtb.soExcited();

}(jQuery));
```

I wanted to log out everyone who was “so excited”. I’m in an object so I can get a reference to the object with this. Then I loop through the object and look for the “isExcited” flag. However, as soon as I enter the loop callback, the context of this has changed and I can’t access the object anymore.

Since jQuery is changing the scope for you in these loops, it’s a good idea to store the reference to this somewhere so that you know it’s not going to change on you.

```
(function($) {

  var sbtb = {
    log: function(message) {
      $("#log").append("").append(message);
    },
    cast: [
      { firstName: "Zack", lastName: "Morris", isExcited: false },
      { firstName: "Kelly", lastName: "Kapowski", isExcited: true
      { firstName: "Lisa", lastName: "Turtle", isExcited: true },
      { firstName: "Screech", lastName: "Powers", isExcited: false
      { firstName: "A.C.", lastName: "Slater", isExcited: false },
      { firstName: "Jessie", lastName: "Spano", isExcited: true },
      { firstName: "Richard", lastName: "Belding", isExcited: fals
    ],
    soExcited: function() {

      // store this in that so we don't get confused later on when
      // our the context of "this" changes out from underneath us
      var that = this;

      // use "that" to get a reference to the cast on this object
      $.each(that.cast, function(idx, actor) {
        // call the log function
        if (actor.isExcited) {
          that.log(actor.firstName + " " + actor.lastName);
          // the value of "that" doesn't change - it's still the o
```

```
        }
      });
    }
  };

  sbtb.soExcited();

}(jQuery));
```

4. Stop Using ALL THE jQUERIES

jQuery has steadily kept increasing in size. This is only natural as new functionality is added. Even though the size has steadily been reduced since 1.8.3, It's led to some decry from the community claiming it "unsuitable" for mobile development due to it's sheer mass.

However, jQuery is not an all or nothing library anymore. jQuery now supports custom builds. I know it's really tempting to use the jQuery CDN and just rock on, but it is important to think about all the code that you are asking your users to download that they might not even need. That's no big deal on a desktop, but bits get precious on mobile devices, and there is no point in sending down a bunch of code to support legacy browsers if your app is a mobile one.

You have two choices here. You can head over to the GitHub site and create a custom build with Git. It's actually really easy and I had no issues getting it to work. But, if pounding out node commands is not your thing, John Resig tweeted about a web UI for custom builds the other day.

5. Stop Using jQuery…

…when you don't need to.

I got to a point in my development career where the first thing that I did with any project was add jQuery, even if I was just creating very simple projects and samples. I did this mainly so that I could just use the DOM selection utilities. Back in the day of older browsers, this was easier to justify, but modern browsers have this whole DOM selection thing already nailed down for you.

DOCUMENT.QUERYSELECTOR

If you are using at least IE version 8 and above, you can just map the $ to document.querySelector, which will return you the first matched element from the selector. You can pass any CSS selector to the function.

Note that IE 8 only supports CSS 2.1 selectors for querySelector.

```
<div class="container">
  <ul>
    <li id="pink">Pink</li>
    <li id="salmon">Salmon</li>
    <li id="blue">Blue</li>
    <li id="green">Green</li>
```

```
    <li id="red">Red</li>
  </ul>
</div>

<script>

  // create a global '$' variable
  window.$ = function(selector) {
    return document.querySelector(selector);
  };

  (function() {
    // select item1 by id and change it's background color to salm
    var item = $("#salmon").style.backgroundColor="salmon";
    console.log(item);
  }());
</script>
```

Try It

Of course, there is no chaining when using the native “style” method, so item logs out “salmon” which is what the backgroundColor method returned.

Having a DOM node is sometimes more handy than having a jQuery wrapped one. For instance, let’s say we want to get a reference to an image and change it’s source. Have a look at how you would do it with jQuery as opposed to a straight up DOM node.

```
// the jQuery way
$("#picture").attr("src", "http://placekitten.com/200/200");

// Vanilla JS with $ mapped to querySelector
$("#picture").src = "http://placekitten.com/200/200";
```

The DOM object gives you direct access to the “src” property of the image because you have a node of type “image” that has access that property. In the case of jQuery, everything is a jQuery object, so you have to go through the attr function to set the image source.

The document.querySelector method only gets you one element. If you call that on a collection of elements, it will return you only the first matched node. You can use document.querySelectorAll to get the entire list.

DOCUMENT.QUERYSELECTORALL

A neat trick is to map $ to querySelector (1 result) and map $$ to querySelectorAll which will give you all matched DOM elements. The tricky part of this is that querySelectorAll returns a node list which isn’t terribly helpful. You are probably going to need this as an array which you can slice up. You can convert the node list to an array by using Array.prototype.slice.call(nodeList).

```
<div class="container">
  <ul>
    <li id="pink">Pink</li>
    <li id="salmon">Salmon</li>
    <li id="blue">Blue</li>
    <li id="green">Green</li>
    <li id="red">Red</li>
  </ul>
</div>

<script>

  // custom lightweight selector implementation
  // nickname: dolla

  window.$ = function(selector) {
    return document.querySelector(selector);
  };

  window.$$ = function(selector) {
    var items = {},
    results = [],
    length = 0,
    i = 0;

    // this doesn't work on IE 8- and Blackberry Browser
    results = Array.prototype.slice.call(document.querySelectorAll

    length = results.length;

    // add the results to the items object
    for ( ; i < length; ) {
      items[i] = results[i];
      i++;
    }

    // add some additional properties to this items object to
    // make it look like an array
    items.length = length;
    items.splice = [].splice();

    // add an 'each' method to the items
    items.each = function(callback) {
      var i = 0;
      for ( ; i < length; ) {
        callback.call(items[i]);
        i++;
```

```
      }
    }

    return items;
  };

  // end custom selector API

  (function() {

    // select the green item and crank up the font size
    $("#green").style.fontSize = "2em";

    // select item1 by id and change it's background color to salm
    $$("li").each(function() {
      this.style.backgroundColor = this.id;
    });
  }());

</script>
```

Try It

Note that converting a nodeList to an array is not supported on IE 8 and below.

At this point you may think to yourself, “That’s a lot of JavaScript to write when I can just include jQuery bro,” and that’s fair. This could be reduced no doubt, but jQuery puts so much convenience on objects and collections that when you start rolling without it, you see what sort of code it takes to recreate just a small piece of it – and this implementation doesn’t even support IE 8 and below or Blackberry browser. Depending on what your application needs to do though, it could be far less code than including jQuery.

Leland Richardson and Jonathan Sampson helped me add a few more features to and clean up the above code. We created a repo called “Dolla“. It’s not meant to be a replacement for jQuery in any way whatsoever, but rather a chance for us to learn what it’s like to say “I don’t need jQuery”. It’s a fair bit more code than you might anticipate.

Stop Reading Articles That Tell You To Stop Doing Something

Seriously though. This is just to underscore the point that these are suggestions, not absolute rules of law. There are a thousand ways that you can argue and or justify any of these items. This is merely to get you thinking about how you use jQuery and how you might be able to do it better than you do now.

So enjoy jQuery, be happy, and stay in school.

Burke Holland

Burke Holland is a web developer who hangs out in Nashville, TN even though he doesn't really care for country music. Previously, he was an Adobe Flex developer and is currently a JavaScript / HTML5 fanatic working as a Developer Evangelist for Kendo UI. You can find him blogging for Kendo UI and on his personal blog A Shiny New Me. He hangs out on twitter as @burkeholland and avoids Facebook altogether. He has an obsession with Instagram and once updated everyone's last name in the corporate ERP database to "Holland". He is not a fan of SQL.

Follow Burke on Twitter

Visit Burke's site