From jQuery to JavaScript: A Reference

Jeffrey Way on Jan 19th 2012 with 81 comments

Tutorial Details

-
- **Subject:** jQuery and JavaScript
- **Difficulty:** Moderate

Whether we like it or not, more and more developers are being introduced to the world of JavaScript through jQuery first. In many ways, these newcomers are the lucky ones. They have access to a plethora of new JavaScript APIs, which make the process of DOM traversal (something that many folks depend on jQuery for) considerably easier. Unfortunately, they don't know about these APIs!

In this article, we'll take a variety of common jQuery tasks, and convert them to both modern and legacy JavaScript.

> **Modern vs. Legacy** – For each item in the list below, you'll find the modern, "cool kids" way to accomplish the task, and the legacy, "make old browsers happy" version. The choice you choose for your own projects will largely depend on your visitors.

Before We Begin

Please note that some of the **legacy** examples in this article will make use of a simple, cross-browser, `addEvent` function. This function will simply ensure that both the W3C-recommended event model, `addEventListener`, and Internet Explorer's legacy `attachEvent` are normalized.

So, when I refer to `addEvent(els, event, handler)` in the legacy code snippets below, the following function is being referenced.

```
var addEvent = (function () {
   var filter = function(el, type, fn) {
      for ( var i = 0, len = el.length; i < len; i++ ) {
         addEvent(el[i], type, fn);
      }
   };
   if ( document.addEventListener ) {
      return function (el, type, fn) {
         if ( el && el.nodeName || el === window ) {
            el.addEventListener(type, fn, false);
         } else if (el && el.length) {
            filter(el, type, fn);
         }
      };
   }

   return function (el, type, fn) {
      if ( el && el.nodeName || el === window ) {
         el.attachEvent('on' + type, function () { return fn.call(el, window.even
      } else if ( el && el.length ) {
         filter(el, type, fn);
```

```
        }
    };
})();

// usage
addEvent( document.getElementsByTagName('a'), 'click', fn);
```

1 – `$('#container');`

This function call will query the DOM for the element with an `id` of `container`, and create a new `jQuery` object.

Modern JavaScript

```
var container = document.querySelector('#container');
```

`querySelector` is part of the Selectors API, which provides us with the ability to query the DOM using the CSS selectors that we're already familiar with.

This particular method will return the first element that matches the passed selector.

Legacy

```
var container = document.getElementById('container');
```

Pay special attention to how you reference the element. When using `getElementById`, you pass the value alone, while, with `querySelector`, a CSS selector is expected.

2 – `$('#container').find('li');`

This time, we're not hunting for a single element; instead, we're capturing any number of list items that are descendants of `#container`.

Modern JavaScript

```
var lis = document.querySelectorAll('#container li');
```

`querySelectorAll` will return *all* elements that match the specified CSS selector.

Selector Limitations

While nearly all relevant browsers support the Selectors API, the specific CSS selectors you pass are still limited to the capability of the browser. Translation: Internet Explorer 8 will only support CSS 2.1 selectors.

Legacy

```
var lis = document.getElementById('container').getElementsByTagName('li');
```

3 – `$('a').on('click', fn);`

In this example, we're attaching a `click` event listener to all anchor tags on the page.

Modern JavaScript

```
[].forEach.call( document.querySelectorAll('a'), function(el) {
   el.addEventListener('click', function() {
     // anchor was clicked
  }, false);

});
```

The above snippet looks scary, but it's not too bad. Because `querySelectorAll` returns a static `NodeList` rather than an `Array`, we can't directly access methods, like `forEach`. This is remedied by calling `forEach` on the `Array` object, and passing the the results of `querySelectorAll` as `this`.

Legacy

```
var anchors = document.getElementsbyTagName('a');
addEvent(anchors, 'click', fn);
```

---

4 – `$('ul').on('click', 'a', fn);`

Ahh – this example is slightly different. This time, the jQuery snippet is using event delegation. The `click` listener is being applied to all unordered lists, however, the callback function will only fire if the target (what the user specifically clicked on) is an anchor tag.

Modern JavaScript

```
document.addEventListener('click', function(e) {
   if ( e.target.matchesSelector('ul a') ) {
      // proceed
   }
}, false);
```

Technically, this vanilla JavaScript method isn't the same as the jQuery example. Instead, it's attaching the event listener directly to the `document`. It then uses the new `matchesSelector` method to determine if the `target` – the node that was clicked – matches the provided selector. This way, we're attaching a single event listener, rather than many.

Please note that, at the time of this writing, all browsers implement `matchesSelector` via their own respective prefixes: `mozMatchesSelector`, `webkitMatchesSelector`, etc. To normalize the method, one might write:

```
var matches;

(function(doc) {
   matches =
      doc.matchesSelector ||
      doc.webkitMatchesSelector ||
      doc.mozMatchesSelector ||
      doc.oMatchesSelector ||
      doc.msMatchesSelector;
})(document.documentElement);

document.addEventListener('click', function(e) {
   if ( matches.call( e.target, 'ul a') ) {
      // proceed
   }
}, false);
```

With this technique, in Webkit, matches will refer to `webkitMatchesSelector`, and, in Mozilla, `mozMatchesSelector`.

Legacy

```
var uls = document.getElementsByTagName('ul');

addEvent(uls, 'click', function() {
   var target = e.target || e.srcElement;
   if ( target && target.nodeName === 'A' ) {
      // proceed
   }
});
```

As a fallback, we determine if the `nodeName` property (the name of the target element) is equal to our desired query. Pay special attention to the fact that older versions of Internet Explorer sometimes plays by their own rules – sort of like the kid who eats play-doh during lunch time. You won't be able to access `target` directly from the `event` object. Instead, you'll want to look for `event.srcElement`.

---

5 - `$('#box').addClass('wrap');`

jQuery provides a helpful API for modifying class names on a set of elements.

Modern JavaScript

```
document.querySelector('#box').classList.add('wrap');
```

This new technique uses the new `classList` API to `add`, `remove`, and `toggle` class names.

```
var container = document.querySelector('#box');

container.classList.add('wrap');
container.classList.remove('wrap');
container.classList.toggle('wrap');
```

Legacy

```
var box = document.getElementById('box'),

    hasClass = function (el, cl) {
        var regex = new RegExp('(?:\\s|^)' + cl + '(?:\\s|$)');
        return !!el.className.match(regex);
    },

    addClass = function (el, cl) {
        el.className += ' ' + cl;
    },

    removeClass = function (el, cl) {
        var regex = new RegExp('(?:\\s|^)' + cl + '(?:\\s|$)');
        el.className = el.className.replace(regex, ' ');
    },

    toggleClass = function (el, cl) {
        hasClass(el, cl) ? removeClass(el, cl) : addClass(el, cl);

    };

addClass(box, 'drago');
removeClass(box, 'drago');
toggleClass(box, 'drago'); // if the element does not have a class of 'drago', ad
```

The fallback technique requires just a tad more work, ay?

---

6 - `$('#list').next();`

jQuery's `next` method will return the element that immediately follows the current element in the wrapped set.

Modern JavaScript

```
var next = document.querySelector('#list').nextElementSibling; // IE9
```

`nextElementSibling` will refer specifically to the next *element* node, rather than any node (text, comment, element). Unfortunately, Internet Explorer 8 and below do not support it.

Legacy

```
var list = document.getElementById('list'),
   next = list.nextSibling;

// we want the next element node...not text.
while ( next.nodeType > 1 ) next = next.nextSibling;
```

There's a couple ways to write this. In this example, we're detecting the `nodeType` of the node that follows the specified element. It could be text, element, or even a comment. As we specifically need the next element, we desire a `nodeType` of `1`. If `next.nodeType` returns a number greater than `1`, we should skip it and keep going, as it's probably a text node.

---

7 - `$('<div id=box></div>').appendTo('body');`

In addition to querying the DOM, jQuery also offers the ability to create and inject elements.

Modern JavaScript

```
var div = document.createElement('div');
div.id = 'box';
document.body.appendChild(div);
```

There's nothing modern about this example; it's how we've accomplished the process of creating and injecting elements into the DOM for a long, long time.

You'll likely need to add content to the element, in which case you can either use `innerHTML`, or `createTextNode`.

```
div.appendChild( document.createTextNode('wacka wacka') );

// or

div.innerHTML = 'wacka wacka';
```

---

8 – `$(document).ready(fn)`

jQuery's `document.ready` method is incredibly convenient. It allows us to begin executing code as soon as possible after the DOM has been loaded.

Modern JavaScript

```
document.addEventListener('DOMContentLoaded', function() {
   // have fun
});
```

Standardized as part of HTML5, the `DOMContentLoaded` event will fire as soon as the document has been completed parsed.

Legacy

```
// http://dustindiaz.com/smallest-domready-ever
function ready(cb) {
   /in/.test(document.readyState) // in = loadINg
      ? setTimeout('ready('+cb+')', 9)
      : cb();
}

ready(function() {
   // grab something from the DOM
});
```

The fallback solution, every nine milliseconds, will detect the value of `document.readyState`. If "loading" is returned, the document hasn't yet been fully parsed (`/in/.test()`. Once it has, though, `document.readyState` will equal "complete," at which point the user's callback function is executed.

---

9 – `$('.box').css('color', 'red');`

If possible, always add a `class` to an element, when you need to provide special styling. However, sometimes, the styling will be determined dynamically, in which case it needs to be inserted as an attribute.

Modern JavaScript

```
[].forEach.call( document.querySelectorAll('.box'), function(el) {
  el.style.color = 'red'; // or add a class
});
```

Once again, we're using the `[].forEach.call()` technique to filter through all of the elements with a class of `box`, and make them red, via the `style` object.

Legacy

```
var box = document.getElementsByClassName('box'), // refer to example #10 below fo
   i = box.length;

while ( i-- > 0 && (box[i].style.color = 'red') );
```

This time, we're getting a bit tricky with the `while` loop. Yes, it's a bit snarky, isn't it? Essentially, we're mimicking:

```
var i = 0, len;

for ( len = box.length; i < len; i++ ) {
   box[i].style.color = 'red';
}
```

However, as we only need to perform a single action, we can save a couple lines. Note that readability is far more important than saving two lines – hence my "snarky" reference. Nonetheless, it's always fun to see how condensed you can make your loops. We're developers; we do this sort of stuff for fun! Anyhow, feel free to stick with the `for` statement version.

---

10 – `$()`

Clearly, our intention is not to replicate the entire jQuery API. Typically, for non-jQuery projects, the `$` or `$$` function is used as shorthand for retrieving one or more elements from the DOM.

Modern JavaScript

```
var $ = function(el) {
   return document.querySelectorAll(el);
};
// Usage = $('.box');
```

Notice that `$` is simply a one-character pointer to `document.querySelector`. It saves time!

Legacy

```
if ( !document.getElementsByClassName ) {
   document.getElementsByClassName = function(cl, tag) {
      var els, matches = [],
         i = 0, len,
         regex = new RegExp('(?:\\s|^)' + cl + '(?:\\s|$)');

      // If no tag name is specified,
      // we have to grab EVERY element from the DOM
      els = document.getElementsByTagName(tag || "*");
      if ( !els[0] ) return false;

      for ( len = els.length; i < len; i++ ) {
         if ( els[i].className.match(regex) ) {
            matches.push( els[i]);
         }
      }
      return matches; // an array of elements that have the desired classname
   };
}

// Very simple implementation. We're only checking for an id, class, or tag name.
// Does not accept CSS selectors in pre-querySelector browsers.
var $ = function(el, tag) {
   var firstChar = el.charAt(0);

   if ( document.querySelectorAll ) return document.querySelectorAll(el);

   switch ( firstChar ) {
      case "#":
         return document.getElementById( el.slice(1) );
      case ".":
         return document.getElementsByClassName( el.slice(1), tag );
      default:
         return document.getElementsByTagName(el);
   }
};

// Usage
$('#container');
$('.box'); // any element with a class of box
$('.box', 'div'); // look for divs with a class of box
$('p'); // get all p elements
```

Unfortunately, the legacy method isn't quite so minimal. Honestly, at this point, you should use a library. jQuery is highly optimized for working with the DOM, which is why it's so popular! The example above will certainly work, however, it doesn't support complex CSS selectors in older browsers; that task is just a wee-bit more complicated!

Summary

It’s important for me to note that that I’m not encouraging you to abandon jQuery. I use it in nearly all of my projects. That said, don’t always be willing to embrace abstractions without taking a bit of time to research the underlying code.

I’d like this posting to serve as a living document, of sorts. If you have any of your own (or improvements/clarifications for my examples), leave a comment below, and I’ll sporadically update this posting with new items. Bookmark this page now! Lastly, I’d like to send a hat-tip to this set of examples, which served as the impetus for this post.

Like Be the first of

By Jeffrey Way
I'm the editor of Nettuts+, and help out on the Envato marketplaces.. I also (very infrequently) blog on my personal site. Feel free to check that out as well...if you're bored. Follow us on Twitter, and subscribe to the official Nettuts+ YouTube page if you want to live.