Open in app Get started

Published in Level Up Coding

Faisal Rashid Follow

Jun 21, 2020 · 6 min read · Listen

Save

JavaScript vs jQuery — Learn DOM manipulation without using jQuery

Originally posted on https://faisalrashid.tech/blogs/JavaScript-vs-jQuery

Open in app Get started

jQuery has been the savior for so many new and coming Web Developers, including myself. If you wanted to learn Web Development back in the day, learning jQuery was an absolute given. This was mainly because jQuery took much of the cross-browser compatibility issues out and enabled developers to write code without having to worry about whether the features that they are implementing will work on all browsers.

But with improvements in browser standards and most of jQuery's API's integrated into JavaScript, jQuery has become a little redundant. Moreover, with native browser API's it is so much easier to debug code, and being native, most of these API's offer better performance than jQuery's API's. Besides, you will have one less library to add to your script imports. If you're still not sold on parting with jQuery maybe this answer will help.

So, if you're considering a move away from jQuery, I have compiled a list of common jQuery methods and API's that people use, with their Vanilla JS alternatives (Vanilla JS is a fancy name for plain JavaScript code without the use of any libraries). Let's dive in!

Querying the DOM

The bread and butter of jQuery is it's amazing ability to query DOM elements. This is demonstrated below:

```
jQuery('div.home')
```

However, you can achieve the same thing with JavaScript using it **document.querySelector()** and **document.querySelectorAll()** methods. Below is their implementation.

```
document.querySelector('div.home')
```

This method always returns the first element that matches the selector. If you're expecting to query multiple elements, you will have to use the **querySelectorAll()** method

Open in app Get started

```
})
```

querySelectoryAll() will return an array of matched elements, hence the use of **forEach** to iterate over each element.

.html()

.html() method of jQuery is used to get or set the HTML content of a node/element.

To do the same in JavaScript you can use the **.innerHtml** property of any HTML element.

Let's say you need to get the HTML content of a div element with class **main-div.** You could do this in JavaScript like this.

```
var htmlContent = document.querySelector('.main-div').innerHtml;
```

Similarly to set the HTML content of the same div, you could do something like this:

```
document.querySelector('.main-div').innerHtml = '<p>Hello World</p>';
```

Remember, if you want to change the Html of multiple elements with the same selector, you can use the **querySelectorAll()** method like this:

```
document.querySelectorAll('.main-div').forEach((el, index) => {
        _el.innerHtml = '<p>Hello World</p>';
});
```

Notice that I pass in the **index** to the anonymous function here. You don't really need it in this case, but it's a good practice to keep the index of each element of the array handy. You never know when you may need it


Open in app Get started

To get or set the value of an input element in JavaScript, you can use the **value** property of that element.

To set the value of an input element with class **input-box** to **'new'**

```
document.querySelector('.input-box').value = 'new';
```

To get the value of the same element

```
var inputValue = document.querySelector('.input-box').value;
```

.text()

jQuery offers **.text()** method to get all the content of an element without the markup. e.g., Using jQuery to get the **text** of Hi element will return the string **'Hi'.** This can be very useful when you're trying to get the content actually visible in the web-page.

That being said, every HTML element has a property "innerText" that essentially holds the same value.

```
let textValue = document.querySelector('.main-div').innerText;
```

Styling the DOM

One of the most exciting features of jQuery is the ability to modify the styling of our DOM. This is highly useful when you're trying to have visual feedback to a user interaction.

Modifying styles can be done by one of two ways:

Adding/Removing Classes:

To add or remove a class you will most definitely use **.addClass()** or **.removeClass()**

 

Open in app Get started

can be done as shown below.
To add the class **testClass** to an element with Id **testElement**

```
document.querySelector('#testElement').classList.add('testClass');
```

To remove the class **testClass** from the element with Id **testElement**

```
document.querySelector('#testElement').classList.remove('testClass')
```

Adding/Modifying CSS:
To directly change the CSS of an element we use the **.css()** method provided by jQuery. Once again, this is fairly simple to do in JavaScript.
To set a style for an element, we simply set the value of that style as exposed by the **style** property of that element.

```
document.querySelector('#testElement').style.color = '#ffffff';
```

This will set the color of the element with Id **testElement** to white.
Note: Since the styles of elements are exposed as JavaScript object properties, you won't be able to use properties such as **background-color** directly as there is a "-" in the property name. In these situations, you should simply use the **camelCased** version of these properties. e.g.,

23

```
document.querySelector('#testElement').style.backgroundColor =
'#000000';
```

This will set the background-color of element with Id **testElement** to black.

Open in app Get started

Setting and retrieving values of the attributes of HTML elements is once again very useful when developing a web application. You may need to get the value of a **data-** attribute that is storing important data while working with your application. jQuery provides this ability by exposing **.attr()** method. You can use it with one parameter to get the value of an attribute or use an overload with two parameters to set the value of an HTML attribute.

In JavaScript, every attribute of an HTML element is exposed as a property of that element. This means that to set the value of the **src** attribute of an **img** element, you can simply do the following:

```
document.querySelector('img').src = '/images/image.png';
```

It's that simple. For attributes such as **data-** you can use another approach,

```
document.querySelector('img').setAttribute('data-title', 'test');
```

To get the value of the same element,

```
document.querySelector('img').getAttribute('data-title');
```

The **getAttribute()** and **setAttribute()** bear a striking resemblance to jQuery's **attr()** method. So, getting used to these shouldn't be that difficult.

.show() / .hide()

Once again, two of the most widely used jQuery methods are the **.show()** and **.hide()** methods. These methods do exactly what they say they do, hide or show HTML elements. In their essence, these methods are simply modifying the display style property of elements. So, we can use simple style modification to achieve the same thing.


Open in app Get started

```
document.querySelector('.test-element').style.display = 'none';
```

To show a hidden element

```
document.querySelector('.test-element').style.display = 'block';
```

AJAX

One of my favorite and most used jQuery functionality is sending HTTP requests using **AJAX.** However, we have to understand that **jQuery.ajax()** is a wrapper around existing JavaScript functionality. There is no doubt that jQuery makes using **AJAX** a breeze, but it's not why we're here, is it?

To send a POST request to the URL 'home/sendmessage' with the data {id: 23, name: 'faisal'}, we'd do the following:

```
let request = new XMLHttpRequest();

// Set an event handler when the status of the request changes

request.onreadystatechange = function(response) {
    if (request.readystate == 4 && request.status == 200) {
        // do stuff with the response object
    }
}

// Open a Post request to the url home/sendmessage asynchronously.

request.open('POST', 'home/sendmessage', true);

// Set the request header/s

request.setRequestHeader('Content-type', 'application/x-www-form-
urlencoded');
```


Open in app Get started

That's it. Your post request will be sent to the server.

If you're a jQuery developer, you will know that these aren't all the methods and features that jQuery ships with. There are definitely more, but I've simply tried to address the most common ones just to give you an idea that life without jQuery in the browser is possible.

I hope that it's enough to push you towards using more and more JavaScript.

Sign up for Top Stories

By Level Up Coding

A monthly summary of the best stories shared in Level Up Coding Take a look.

Get this newsletter

About Help Terms Privacy

Get the Medium app

Download on the App Store GET IT ON Google Play