Pocket Guides Go Make Things

4

DOM INJECTION & TRAVERSAL

with Vanilla JavaScript

CHRIS FERDINANDI

DOM Injection and Traversal

By Chris Ferdinandi

Go Make Things, LLC

v2.2.0

Table of Contents

Intro

In this guide, you'll learn:

- How to detect when the viewport is ready.
- How to manipulation HTML.
- How to add and remove elements from the DOM.
- How to traverse up and down the DOM.
- How to detect when elements are in the viewport.
- How to calculate distances in the viewport.

A quick word about browser compatibility

This guide makes heavy use of ECMAScript 5 (more commonly known as ES5) and ECMA 6 (ES6) methods and APIs.

My goal for browser support is IE9 and above. Each function or technique mentioned in this guide includes specific browser support information. For methods and APIs that don't meet that standard, I also include information about polyfills—snippets of code that add support for features to browsers that don't natively offer it.

You'll never have to run a command line prompt, compile code, or learn a weird pseudo language (though you certain can if you want to).

***Note:** You can extend support all the way back to IE7 with a polyfill service like polyfill.io*[1].

Using the code in this guide

Let’s get started!

DOM Ready

How to detect when the DOM is ready before running code.

Note: *If you're loading your scripts in the footer (which you generally should be for performance reasons), the `ready()` method isn't really needed. It's just a habit from the "load everything in the header" days.*

Vanilla JavaScript provides a native way to do this: the `DOMContentLoaded` event for `addEventListener`.

But… if the DOM is already loaded by the time you call your event listener, the event never happens and your function never runs.

`ready()` is a lightweight helper method[2] that does the same thing as jQuery's `ready()` method. It does two things:

1. Check to see if the document is already `interactive` or `complete`. If so, it runs your function immediately.
2. Otherwise, it adds a listener for the `DOMContentLoaded` event.

```
/*!
 * Run event after DOM is ready
 * (c) 2017 Chris Ferdinandi, MIT License, https://gomakethings.com
 * @param  {Function} fn Callback function
 */
var ready = function (fn) {

    // Sanity check
    if (typeof fn !== 'function') return;

    // If document is already loaded, run method
    if (document.readyState === 'interactive' || document.readyState === 'complete') {
        return fn();
    }

    // Otherwise, wait until document is loaded
    document.addEventListener('DOMContentLoaded', fn, false);

};

// Example
ready(function() {
    // Do stuff...
});
```

Browser Compatibility

Works in all modern browsers, and IE9 and above.

HTML and Content

innerHTML

Use `innerHTML` to get and set HTML content in an element.

```
var elem = document.querySelector('#some-elem');

// Get HTML content
var html = elem.innerHTML;

// Set HTML content
elem.innerHTML = 'We can dynamically change the HTML. We can even include HTML elements like <a href="#">this link</a>.';

// Add HTML to the end of an element's existing content
elem.innerHTML += ' Add this after what is already there.';

// Add HTML to the beginning of an element's existing content
elem.innerHTML = 'We can add this to the beginning. ' + elem.innerHTML;

// You can inject entire elements into other ones, too
elem.innerHTML += '<p>A new paragraph</p>';
```

Browser Compatibility

Works in all modern browsers, and IE9 and above. **IE9 Exception:** Tables and selects require IE10 and above.[3]

textContent

The `textContent` property works just like `innerHTML`, but only gets the text of an element and omits the markup.

```
var elem = document.querySelector('#some-elem');

// Get text content
var text = elem.textContent;

// Set text content
elem.textContent = 'We can dynamically change the content.';

// Add text to the end of an element's existing content
elem.textContent += ' Add this after what is already there.';

// Add text to the beginning of an element's existing content
elem.textContent = 'We can add this to the beginning. ' + elem.textContent;
```

Browser Compatibility

Works in all modern browsers, and IE9 and above.

DOM Injection

How to add and remove elements in the DOM.

createElement()

Use the `document.createElement()` method to create an element.

```
var div = document.createElement('div');
var link = document.createElement('a');
var article = document.createElement('article');
```

You can use an valid HTML tag, and even create invalid ones, too. For example, these also work.

```
var chicken = document.createElement('chicken'); // <chic
ken></chicken>
var placeholder = document.createElement('_'); // <_></_>
```

You can manipulate an element created with `createElement()` like you would any other element in the DOM. Add classes, attributes, styles, and more.

```
var div = document.createElement('div');
div.className = 'new-div';
div.id = 'new-div';
div.setAttribute('data-div', 'new');
div.style.color = '#fff';
div.style.backgroundColor = 'rebeccapurple';
```

Browser Compatibility

Works in all modern browsers, and at least IE6.

insertBefore()

The `insertBefore()` method is the original way to add elements to the DOM. Call it on the parent of the element you're inserting your new element before (the `referenceNode`), and pass in both the new element and the reference node as arguments.

```
// Create a new element
var newNode = document.createElement('div');

// Get the reference node
var referenceNode = document.getElementById('some-element');

// Insert the new node before the reference node
referenceNode.parentNode.insertBefore(newNode, referenceNode);
```

Oddly, you can also use it to inject an element *after* another one by using the `.nextSibling` property on your `referenceNode`.

```
// Create a new element
var newNode = document.createElement('div');

// Get the reference node
var referenceNode = document.getElementById('#some-element');

// Insert the new node after the reference node
referenceNode.parentNode.insertBefore(newNode, referenceNode.nextSibling);
```

Browser Compatibility

Works in all modern browsers, and at least IE6.

before()

The `before()` method is a new ES6 function that let's you insert an element before another one. Call the `before()` method on the reference node, and pass in the new node as an argument.

```
// Create a new element
var newNode = document.createElement('div');

// Get the reference node
var referenceNode = document.querySelector('#some-element
');

// Insert the new node before the reference node
referenceNode.before(newNode);
```

Browser Compatibility

Works in newer versions of Chrome, Firefox, and Opera, and has no support in Edge and Internet Explorer. Support can be added back to IE9 with a polyfill[4].

```
/**
 * ChildNode.before() polyfill
 */
```

```
// from: https://github.com/jserz/js_piece/blob/master/DO
M/ChildNode/before()/before().md
(function (arr) {
    arr.forEach(function (item) {
        if (item.hasOwnProperty('before')) {
            return;
        }
        Object.defineProperty(item, 'before', {
            configurable: true,
            enumerable: true,
            writable: true,
            value: function before() {
                var argArr = Array.prototype.slice.call(a
rguments),
                    docFrag = document.createDocumentFrag
ment();

                argArr.forEach(function (argItem) {
                    var isNode = argItem instanceof Node;
                    docFrag.appendChild(isNode ? argItem
: document.createTextNode(String(argItem)));
                });

                this.parentNode.insertBefore(docFrag, thi
s);
            }
        });
    });
})([Element.prototype, CharacterData.prototype, DocumentT
ype.prototype]);
```

after()

The `after()` method is another new ES6 function that inserts an element in the DOM after another one. Call the `after()` method on the reference node, and pass in the new node as an argument.

```
// Create a new element
var newNode = document.createElement('div');

// Get the reference node
var referenceNode = document.querySelector('#some-element
');

// Insert the new node after the reference node
referenceNode.after(newNode);
```

Browser Compatibility

Works in newer versions of Chrome, Firefox, and Opera, and has no support in Edge and Internet Explorer. Support can be added back to IE9 with a polyfill[5].

```
/**
 * ChildNode.after() polyfill
 */
```

```
//from: https://github.com/jserz/js_piece/blob/master/DOM
/ChildNode/after()/after().md
(function (arr) {
    arr.forEach(function (item) {
        if (item.hasOwnProperty('after')) {
            return;
        }
        Object.defineProperty(item, 'after', {
            configurable: true,
            enumerable: true,
            writable: true,
            value: function after() {
                var argArr = Array.prototype.slice.call(a
rguments),
                    docFrag = document.createDocumentFrag
ment();

                argArr.forEach(function (argItem) {
                    var isNode = argItem instanceof Node;
                    docFrag.appendChild(isNode ? argItem
: document.createTextNode(String(argItem)));
                });

                this.parentNode.insertBefore(docFrag, thi
s.nextSibling);
            }
        });
    });
})([Element.prototype, CharacterData.prototype, DocumentT
ype.prototype]);
ype.prototype]);
```

prepend()

The `prepend()` method is another new ES6 function that inserts an element at the beginning of a set elements inside a shared parent. Call the `prepend()` method on the reference node, and pass in the new node as an argument.

```
// Create a new element
var newNode = document.createElement('div');

// Get the parent node
var referenceNode = document.querySelector('#some-element
');

// Insert the new node before the first element in the re
ference node
referenceNode.prepend(newNode);
```

Browser Compatibility

Works in newer versions of Chrome, Firefox, and Opera, and has no support in Edge and Internet Explorer. Support can be added back to IE7 with a polyfill[6].

```
/**
```

```
/
 * ParentNode.prepend() polyfill
 */
// Source: https://github.com/Financial-Times/polyfill-se
rvice
var _mutation = (function () {

    function isNode(object) {
        // DOM, Level2
        if (typeof Node === 'function') {
            return object instanceof Node;
        }
        // Older browsers, check if it looks like a Node
instance)
        return object &&
            typeof object === "object" &&
            object.nodeName &&
            object.nodeType >= 1 &&
            object.nodeType <= 12;
    }

    // http://dom.spec.whatwg.org/#mutation-method-macro
    return function mutation(nodes) {
        if (nodes.length === 1) {
            return isNode(nodes[0]) ? nodes[0] : document
.createTextNode(nodes[0] + '');
        }

        var fragment = document.createDocumentFragment();
        for (var i = 0; i < nodes.length; i++) {
            fragment.appendChild(isNode(nodes[i]) ? nodes
[i] : document.createTextNode(nodes[i] + ''));

        }
        return fragment;
    };
}());

Document.prototype.prepend = Element.prototype.prepend =
function prepend() {
    this.insertBefore(_mutation(arguments), this.firstChi
ld);
};
```

append()

The `append()` method is yet another new ES6 function that inserts an element at the end of a set elements inside a shared parent. Call the `append()` method on the reference node, and pass in the new node as an argument.

```
// Create a new element
var newNode = document.createElement('div');

// Get the reference node
var referenceNode = document.querySelector('#some-element
');

// Insert the new node after the last element in the refe
rence node
referenceNode.append(newNode);
```

Browser Compatibility

Works in newer versions of Chrome, Firefox, and Opera, and has no support in Edge and Internet Explorer. Support can be added back to IE9 with a polyfill[7].

```
/**
 * ParentNode.append() polyfill
 */
// Source: https://github.com/jserz/js_piece/blob/master/
DOM/ParentNode/append()/append().md
(function (arr) {
    arr.forEach(function (item) {
        if (item.hasOwnProperty('append')) {
            return;
        }
        Object.defineProperty(item, 'append', {
            configurable: true,
            enumerable: true,
            writable: true,
            value: function append() {
                var argArr = Array.prototype.slice.call(a
rguments),
                    docFrag = document.createDocumentFrag
ment();

                argArr.forEach(function (argItem) {
                    var isNode = argItem instanceof Node;
                    docFrag.appendChild(isNode ? argItem
: document.createTextNode(String(argItem)));
                });

                this.appendChild(docFrag);
            }
        });
    });
})([Element.prototype, Document.prototype, DocumentFragme
nt.prototype]);
```

appendChild()

The `appendChild()` method is an older function that inserts an element at the end of a set of elements inside a shared parent. Call the `appendChild()` method on the reference node, and pass in the new node as an argument.

```
// Create a new element
var newNode = document.createElement('div');

// Get the reference node
var referenceNode = document.querySelector('#some-element
');

// Insert the new node after the last element in the pare
nt node
referenceNode.appendChild(newNode);
```

Browser Compatibility

Works in all modern browsers, and at least IE6.

Given that it has the same simple structure as `append()`, but much better browser support without a polyfill, I would personally use this method every time.

removeChild()

Use the `removeChild()` method to remove an element from the DOM. This method is called against our target element's parent, which we can get with `parentNode`.

```
var elem = document.querySelector('#some-element');
elem.parentNode.removeChild(elem);
```

Browser Compatibility

Works in all modern browsers, and at least IE6.

remove()

The `remove()` method is, of course, another new ES6 method that let’s you remove an element from the DOM. Call the `remove()` method on the element you want to remove.

```
var elem = document.querySelector('#some-element');
elem.remove();
```

Browser Compatibility

Works in all modern browsers, but not IE. You can add support back to IE9 with a polyfill.[8]

```
/**
 * ChildNode.remove() polyfill
 */
// from:https://github.com/jserz/js_piece/blob/master/DOM
/ChildNode/remove()/remove().md
(function (arr) {
    arr.forEach(function (item) {
        if (item.hasOwnProperty('remove')) {
            return;
        }
        Object.defineProperty(item, 'remove', {
            configurable: true,
            enumerable: true,
            writable: true,
            value: function remove() {
                this.parentNode.removeChild(this);
            }
        });
    });
})([Element.prototype, CharacterData.prototype, DocumentT
ype.prototype]);
```

Traversing Up the DOM

How to traverse up the DOM.

parentNode

Use `parentNode` to get the parent of an element.

```
var elem = document.querySelector('#some-elem');
var parent = elem.parentNode;
```

You can also string them together to go several levels up.

```
var levelUpParent = elem.parentNode.parentNode;
```

Browser Compatibility

Works in all modern browsers, and at least back to IE6.

closest()

Use `closest()` to get the closest parent up the DOM tree that matches against a selector.

```
var elem = document.querySelector('#some-elem');
var closestSandwich = elem.closest('[data-sandwich]');
```

Note: *This method can also be used in event listeners to determine if the `event.target` is inside of a particular element or not (for example, did a click happen inside of a dropdown menu?).*

Browser Compatibility

The `closest()` method works in most modern browsers but has pretty spotty backwards compatibility. Fortunately, a lightweight polyfill[9] adds functionality to IE9 and up.

```
/**
 * Element.closest() polyfill
 * https://developer.mozilla.org/en-US/docs/Web/API/Element/closest#Polyfill
 */
if (!Element.prototype.closest) {
    if (!Element.prototype.matches) {
        Element.prototype.matches = Element.prototype.msMatchesSelector || Element.prototype.webkitMatchesSelector;
    }
    Element.prototype.closest = function (s) {
        var el = this;
        var ancestor = this;
        if (!document.documentElement.contains(el)) return null;
        do {
            if (ancestor.matches(s)) return ancestor;
            ancestor = ancestor.parentElement;
        } while (ancestor !== null);
        return null;
    };
}
```

getParents()

`getParents()` is a helper method[10] I wrote that returns an array of parent elements, optionally matching against a selector. It's a vanilla JavaScript equivalent to jQuery's `.parents()` method.

It starts with the element you've passed in itself, so pass in `elem.parentNode` to skip to the first parent element instead. Here's how to use it.

```
var elem = document.querySelector('#some-elem');
var parents = getParents(elem.parentNode);
var parentsWithWrapper = getParents(elem.parentNode, '.wr
apper');
```

And here's the helper method.

```
/*!
 * Get all of an element's parent elements up the DOM tre
e
 * (c) 2017 Chris Ferdinandi, MIT License, https://gomake
things.com
 * @param  {Node}   elem     The element
 * @param  {String} selector Selector to match against [o
ptional]
 * @return {Array}           The parent elements
 */
var getParents = function (elem, selector) {

    // Element.matches() polyfill
    if (!Element.prototype.matches) {
        Element.prototype.matches =
            Element.prototype.matchesSelector ||
            Element.prototype.mozMatchesSelector ||
            Element.prototype.msMatchesSelector ||
            Element.prototype.oMatchesSelector ||
            Element.prototype.webkitMatchesSelector ||
            function(s) {
                var matches = (this.document || this.owne
rDocument).querySelectorAll(s),
                    i = matches.length;
                while (--i >= 0 && matches.item(i) !== th
is) {}
                return i > -1;
            };
    }

    // Setup parents array
    var parents = [];

    // Get matching parent elements
    for (; elem && elem !== document; elem = elem.parentN
ode) {

        // Add matching parents to array
        if (selector) {
            if (elem.matches(selector)) {
                parents.push(elem);
            }
        } else {
            parents.push(elem);
        }
    }

    return parents;

};
```

Browser Compatibility

Works in all modern browsers, and IE9 and above.

getParentsUntil()

`getParentsUntil()` is a helper method[11] I wrote that returns an array of parent elements until a matching parent is found, optionally matching against a selector. It's a vanilla JavaScript equivalent to jQuery's `.parentsUntil()` method.

It starts with the element you've passed in itself, so pass in `elem.parentNode` to skip to the first parent element instead. Here's how to use it.

```
var elem = document.querySelector('#some-element');
var parentsUntil = getParentsUntil(elem, '.some-class');
var parentsUntilByFilter = getParentsUntil(elem, '.some-class', '[data-something]');
var allParentsUntil = getParentsUntil(elem);
var allParentsExcludingElem = getParentsUntil(elem.parentNode);
```

And here’s the helper method.

```
/*!
 * Get all of an element's parent elements up the DOM tree until a matching parent is found
 * (c) 2017 Chris Ferdinandi, MIT License, https://gomakethings.com
 * @param  {Node}   elem     The element
 * @param  {String} parent   The selector for the parent to stop at
 * @param  {String} selector The selector to filter against [optionals]
 * @return {Array}           The parent elements
 */
var getParentsUntil = function (elem, parent, selector) {

    // Element.matches() polyfill
    if (!Element.prototype.matches) {
        Element.prototype.matches =
            Element.prototype.matchesSelector ||
            Element.prototype.mozMatchesSelector ||
            Element.prototype.msMatchesSelector ||
            Element.prototype.oMatchesSelector ||
            Element.prototype.webkitMatchesSelector ||
            function(s) {
                var matches = (this.document || this.owne
rDocument).querySelectorAll(s),
                    i = matches.length;
                while (--i >= 0 && matches.item(i) !== th
is) {}
                return i > -1;
            };
    }

    // Setup parents array
    var parents = [];

    // Get matching parent elements
    for (; elem && elem !== document; elem = elem.parentN
ode) {

        if (parent) {
            if (elem.matches(parent)) break;
        }

        if (selector) {
            if (elem.matches(selector)) {
                parents.push(elem);
            }
            break;
        }
        }
        parents.push(elem);
    }
    return parents;
};
```

Browser Compatibility

Works in all modern browsers, and IE9 and above.

Traversing Down the DOM

querySelector() and querySelectorAll()

The `querySelector()` and `querySelectorAll()` APIs aren't limited to just running on the `document`. They can be run on any element to search only for elements inside of it.

```
var elem = document.querySelector('#some-elem');

// Find the first element inside `#some-elem` that has a
`[data-snack]` attribute
var snack = elem.querySelector('[data-snack]');

// Get all divs inside `#some-elem`
var divs = elem.querySelectorAll('div');
```

Browser Compatibility

Same as `querySelectorAll` and `querySelector`.

children

While `querySelector()` and `querySelectorAll()` search through all levels within a nested DOM/HTML structure, you may want to just get immediate decedants of a particular element. Use `children` for this.

```
var elem = document.querySelector('#some-elem');
var decendants = wrapper.children;
```

Browser Compatibility

Works in all modern browsers, and IE9 and above.

Traversing Sideways in the DOM

`getSiblings()` is a helper method[12] I wrote that gets the siblings of an element in the DOM. For example: if you had a list item (`<li>`) and wanted to grab all of the other items in the list.

Here’s how to use it.

```
var elem = document.querySelector('#some-element');
var siblings = getSiblings(elem);
```

And here’s the helper method.

```
/*!
 * Get all siblings of an element
 * (c) 2017 Chris Ferdinandi, MIT License, https://gomake
things.com
 * @param  {Node}  elem The element
 * @return {Array}      The siblings
 */
var getSiblings = function (elem) {
    var siblings = [];
    var sibling = elem.parentNode.firstChild;
    for (; sibling; sibling = sibling.nextSibling) {
        if (sibling.nodeType === 1 && sibling !== elem) {
            siblings.push(sibling);
        }
    }
    return siblings;
};
```

Browser Compatibility

Works in all modern browsers, and IE6 and above.

The Viewport

How to detect viewport dimensions, and check if an element is within it.

Get the viewport height

There are two methods to get the viewport height: `window.innerHeight` and `document.documentElement.clientHeight`. The former is more accurate. The latter has better browser support.

To get the best of both worlds, try `innerHeight` first, and fallback to `clientHeight` if not supported.

```
var viewportHeight = window.innerHeight || document.docum
entElement.clientHeight;
```

Browser Compatibility

`innerHeight` works in all modern browsers, and IE9 and above. `clientHeight` works in all modern browsers, and IE6 and above.

Get the viewport width

There are two methods to get the viewport width: `window.innerWidth` and `document.documentElement.clientWidth`. The former is more accurate. The latter has better browser support.

To get the best of both worlds, try `innerWidth` first, and fallback to `clientWidth` if not supported.

```
var viewportWidth = window.innerWidth || document.documen
tElement.clientWidth;
```

Browser Compatibility

`innerWidth` works in all modern browsers, and IE9 and above. `clientWidth` works in all modern browsers, and IE6 and above.

Check if an element is in the viewport or not

`isInViewport()` is a helper method[13] I wrote to check if an element is in the viewport or not. It returns `true` if the element is in the viewport, and `false` if it's not.

Here's how you use it.

```
var elem = document.querySelector('#some-element');
isInViewport(elem); // Boolean: returns true/false
```

And here's the helper method.

```
/*!
 * Determine if an element is in the viewport
 * (c) 2017 Chris Ferdinandi, MIT License, https://gomake
things.com
 * @param  {Node}    elem The element
 * @return {Boolean}      Returns true if element is in t
he viewport
 */
var isInViewport = function (elem) {
    var distance = elem.getBoundingClientRect();
    return (
        distance.top >= 0 &&
        distance.left >= 0 &&
        distance.bottom <= (window.innerHeight || documen
t.documentElement.clientHeight) &&
        distance.right <= (window.innerWidth || document.
documentElement.clientWidth)
    );
};
```

Browser Compatibility

Works in all modern browsers, and IE9 and above.

Distances

How to get the distances between elements in the DOM.

Get the currently scrolled distance from the top of the page

Use `pageYOffset` to get the distance the user has scrolled from the top of the page.

```
var distance = window.pageYOffset;
```

Browser Compatibility

Works in all modern browsers, and IE9 and above.

Get an element’s distance from the top of the page

`getOffsetTop()` is a helper method[14] I wrote to get an element’s distance from the top of the document.

Here’s how it works.

```
var elem = document.querySelector('#some-element');
var distance = getOffsetTop(elem);
```

And here’s the helper method.

```
/*!
 * Get an element's distance from the top of the Document
.
 * (c) 2017 Chris Ferdinandi, MIT License, https://gomake
things.com
 * @param  {Node} elem The element
 * @return {Number}    Distance from the top in pixels
 */
var getOffsetTop = function (elem) {
    var location = 0;
    if (elem.offsetParent) {
        do {
            location += elem.offsetTop;
            elem = elem.offsetParent;
        } while (elem);
    }
    return location >= 0 ? location : 0;
};
```

Browser Compatibility

Works in all modern browsers, and IE9 and above.

Get the distance between two elements in the DOM

We can use `getOffsetTop` to calculate the distance from the top of the viewport for our two elements, and then use simple match to figure out how far apart they are.

```
var elem1 = document.querySelector('#first-element');
var elem2 = document.querySelector('#second-element');

// Distance between them
// If negative, elem1 is below elem2
var distance = getOffsetTop(elem2) - getOffsetTop(elem1);

// To always get a positive number
var distancePositive = Math.abs(getOffsetTop(elem2) - get
OffsetTop(elem1));
```

Putting it all together

To make this all tangible, let's work on a project together. We'll build a script that lazy loads images after they enter the viewport.

The starter template and complete project code are included in the source code[15] on GitHub.

Getting Setup

I've dropped some placeholder markup into the template to help you get started.

Throughout the page copy, I've included empty `<figure>` elements that will eventually contain our lazy loaded images. Each one has the `.lazy-load` class on it, which we'll use as a hook in our JavaScript to get the images that should be lazy loaded.

We also need to include some information about the image itself, which we can do with data attributes. The `[data-image]` attribute includes a URL to the image to be lazy loaded, and the `[data-caption]` attribute (if included) is the caption we'd like to include with our image.

```
<figure class="lazy-load" data-caption="A lifeguard station on a deserted beach" data-image="img/beach.jpg"></figure>
```

I've also included some lightweight CSS to make our images responsive, and our captions look good.

```
figure {
    margin: 0 0 1.4em;
}

caption {
    color: #808080;
    display: block;
    font-size: 0.8em;
    font-style: italic;
    padding: 0.25em 0 0.5em;
    text-align: center;
}

img {
    height: auto;
    max-width: 100%;
}
```

Alright, let’s get coding!

Getting all of the images

The first thing we need to do is get all of the images that we want to lazy load. We’ll use `querySelectorAll()` for that.

```
var images = document.querySelectorAll('.lazy-load');
```

Whenever the visitor scrolls, we want to loop through each of our images, and check to see if it's in the viewport. We'll use `addEventListener()` to listen for scroll events, with a simple `for` loop to loop through each of our image placeholders.

```
var images = document.querySelectorAll('.lazy-load');

window.addEventListener('scroll', function (event) {
    for (var i = 0; i < images.length; i++) {
        // Check if image is in viewport
    }
}, false);
```

To check if our image is in the viewport, we'll use the `isInViewport()` helper method I shared earlier in this guide.

```
// Get all lazy load images
var images = document.querySelectorAll('.lazy-load');

// Determine if an element is in the viewport
var isInViewport = function (elem) {
    var distance = elem.getBoundingClientRect();
    return (
        distance.top >= 0 &&
        distance.left >= 0 &&
        distance.bottom <= (window.innerHeight || documen
t.documentElement.clientHeight) &&
        distance.right <= (window.innerWidth || document.
documentElement.clientWidth)
    );
};

// Listen for scroll events
window.addEventListener('scroll', function (event) {

    // Loop through each lazy load image
    for (var i = 0; i < images.length; i++) {

        // Check if the image is in the viewport
        if (isInViewport(images[i])) {
            // Load the image
            console.log(images[i]);
        }

    }

}, false);
```

If you reload the page, open up the console in dev tools, and scroll, you'll see our image placeholders showing up as the come into the viewport.

Loading the image

If the image is in the viewport, we want to load it into the DOM. First, though, let's make sure our image has a `[data-image]` attribute on it. If not, we can just quit and move on to the next image in our loop.

```
// Listen for scroll events
window.addEventListener('scroll', function (event) {

    // Loop through each lazy load image
    for (var i = 0; i < images.length; i++) {

        // Check if the image is in the viewport
        if (isInViewport(images[i])) {

            // Make sure the image has a data-image attribute
            if (!images[i].hasAttribute('data-image'))) continue;

        }

    }

}, false);
```

To load the image, we can use `innerHTML` to inject it into the placeholder `<figure>` element.

```
// Listen for scroll events
window.addEventListener('scroll', function (event) {

    // Loop through each lazy load image
    for (var i = 0; i < images.length; i++) {

        // Check if the image is in the viewport
        if (isInViewport(images[i])) {

            // Make sure the image has a data-image attri
bute
            if (!images[i].hasAttribute('data-image')) co
ntinue;

            // Load the image
            images[i].innerHTML = '<img src="' + images[i
].getAttribute('data-image') + '">';

        }

    }

}, false);
```

Loading the caption

Next, let’s check if there’s a caption included, and load that, too.

We'll use `hasAttribute()` to see if a caption is included. If it is, we'll climb down the DOM to find our image. Then, we'll create a `<caption>` element and and use `insertBefore` to add it after the image.

Note: *This is not the most efficient way to do this. A better way would be to add the image and caption at the same time using `innerHTML`. But, I wanted to make sure you got to work with as many of the techniques covered in this guide as possible.*

```
// Listen for scroll events
window.addEventListener('scroll', function (event) {

    // Loop through each lazy load image
    for (var i = 0; i < images.length; i++) {

        // Check if the image is in the viewport
        if (isInViewport(images[i])) {

            // Make sure the image has a data-image attri
bute
            if (!images[i].hasAttribute('data-image')) co
ntinue;

            // Load the image
            images[i].innerHTML = '<img src="' + images[i
].getAttribute('data-image') + '">';

            // Add a caption if one exists
            if (images[i].hasAttribute('data-caption')) {
                var img = images[i].querySelector('img');
                var caption = document.createElement('caption');
                caption.innerHTML = images[i].getAttribute('data-caption');
                images[i].insertBefore(caption, img.nextSibling);
            }

        }

    }

}, false);
```

Only load each image once

There’s a small problem with our code. Right now, it reloads the image and caption over and over again as you scroll through a page.

You can see this in action by logging our image in the console.

```
// Listen for scroll events
window.addEventListener('scroll', function (event) {

    // Loop through each lazy load image
    for (var i = 0; i < images.length; i++) {

        // Check if the image is in the viewport
        if (isInViewport(images[i])) {

            // Make sure the image has a data-image attri
bute
            if (!images[i].hasAttribute('data-image')) co
ntinue;

            // Load the image
            images[i].innerHTML = '<img src="' + images[i
].getAttribute('data-image') + '">';

            // Add a caption if one exists
            if (images[i].hasAttribute('data-caption')) {
                var img = images[i].querySelector('img');
                var caption = document.createElement('cap
tion');
                caption.innerHTML = images[i].getAttribut
e('data-caption');
                images[i].insertBefore(caption, img.nextS
ibling);
            }

            // Log the image load in the console
            console.log(images[i]);

        }

    }
```

```
}, false);
```

See how the same images are showing up multiple times? We don’t want that. Let’s add a `.loaded` class to our placeholder after we load the image.

```
// Listen for scroll events
window.addEventListener('scroll', function (event) {

    // Loop through each lazy load image
    for (var i = 0; i < images.length; i++) {

        // Check if the image is in the viewport
        if (isInViewport(images[i])) {

            // Make sure the image has a data-image attri
bute
            if (!images[i].hasAttribute('data-image')) co
ntinue;

            // Load the image
            images[i].innerHTML = '<img src="' + images[i
].getAttribute('data-image') + '">';

            // Add a caption if one exists
            if (images[i].hasAttribute('data-caption')) {
                var img = images[i].querySelector('img');
                var caption = document.createElement('cap
tion');
                caption.innerHTML = images[i].getAttribut
e('data-caption');
                images[i].insertBefore(caption, img.nextS
ibling);
            }

            // Add a .loaded class to our image placehold
er
            images[i].classList.add('loaded');

        }

    }

}, false);
```

Now, where we check to make sure our placeholder has a `[data-image]` attribute, we can also check to see if it has the `.loaded` class.

```
// Listen for scroll events
window.addEventListener('scroll', function (event) {

    // Loop through each lazy load image
    for (var i = 0; i < images.length; i++) {

        // Check if the image is in the viewport
        if (isInViewport(images[i])) {

            // Make sure the image has a data-image attri
bute and hasn't already been loaded
            if (!images[i].hasAttribute('data-image') ||
images[i].classList.contains('loaded')) continue;

            // Load the image
            images[i].innerHTML = '<img src="' + images[i
].getAttribute('data-image') + '">';

            // Add a caption if one exists
            if (images[i].hasAttribute('data-caption')) {
                var img = images[i].querySelector('img');
                var caption = document.createElement('cap
tion');
                caption.innerHTML = images[i].getAttribut
e('data-caption');
                images[i].insertBefore(caption, img.nextS
ibling);
            }

            // Add a .loaded class to our image placehold
er
            images[i].classList.add('loaded');

        }

    }

}, false);
```

Loading images on page load

One last thing.

If an image is above the fold on page load, but the visitor hasn't scrolled yet, it won't load. We want to run the same loop we use on scroll when the page is first loaded.

First, let's pull it out into its own function so that we're not writing the same code twice.

```
// Load our images
var loadImages = function () {

    // Loop through each lazy load image
    for (var i = 0; i < images.length; i++) {

        // Check if the image is in the viewport
        if (isInViewport(images[i])) {

            // Make sure the image has a data-image attri
bute and hasn't already been loaded
            if (!images[i].hasAttribute('data-image') ||
images[i].classList.contains('loaded')) continue;

            // Load the image
            images[i].innerHTML = '<img src="' + images[i
].getAttribute('data-image') + '">';

            // Add a caption if one exists
            if (images[i].hasAttribute('data-caption')) {
                var img = images[i].querySelector('img');
                var caption = document.createElement('cap
tion');
                caption.innerHTML = images[i].getAttribut
e('data-caption');
                images[i].insertBefore(caption, img.nextS
ibling);
            }

            // Add a .loaded class to our image placehold
er
            images[i].classList.add('loaded');

        }

    }

};
```

Then, we can call that function in our event listener.

```
// Listen for scroll events
window.addEventListener('scroll', loadImages, false);
```

Now, let’s also call that function the first time our script runs.

```
// Listen for scroll events
window.addEventListener('scroll', loadImages, false);

// Load images on page load
loadImages();
```

Now, any images in the viewport on page load will show up instantly. The rest will load when they're scrolled into the viewport.

Congratulations! You just created an image lazy loader using a handful of DOM injection and traversal techniques.

About the Author

Hi, I'm Chris Ferdinandi. I help people learn JavaScript.

I love pirates, puppies, and Pixar movies, and live near horse farms in rural Massachusetts. I run Go Make Things with Bailey Puppy, a lab-mix from Tennessee.

You can find me:

- On my website at GoMakeThings.com.
- By email at chris@gomakethings.com.
- On Twitter at @ChrisFerdinandi.

1. https://polyfill.io↩
2. https://github.com/cferdinandi/vanilla-javascript-cheat-sheet/blob/master/helper-methods/ready.js↩
3. http://quirksmode.org/dom/html/↩
4. https://github.com/cferdinandi/vanilla-javascript-cheat-sheet/blob/master/polyfills/before.js↩
5. https://github.com/cferdinandi/vanilla-javascript-cheat-sheet/blob/master/polyfills/after.js↩
6. https://github.com/cferdinandi/vanilla-javascript-cheat-sheet/blob/master/polyfills/prepend.js↩
7. https://github.com/cferdinandi/vanilla-javascript-cheat-sheet/blob/master/polyfills/append.js↩
8. https://github.com/cferdinandi/vanilla-javascript-cheat-sheet/blob/master/polyfills/remove.js↩
9. https://github.com/cferdinandi/vanilla-javascript-cheat-sheet/blob/master/polyfills/closest.js↩
10. https://github.com/cferdinandi/vanilla-javascript-cheat-sheet/blob/master/helper-methods/getParents.js↩
11. https://github.com/cferdinandi/vanilla-javascript-cheat-sheet/blob/master/helper-methods/getParentsUntil.js↩
12. https://github.com/cferdinandi/vanilla-javascript-cheat-sheet/blob/master/helper-methods/getSiblings.js↩
13. https://github.com/cferdinandi/vanilla-javascript-cheat-sheet/blob/master/helper-methods/isInViewport.js↩
14. https://github.com/cferdinandi/vanilla-javascript-cheat-sheet/blob/master/helper-methods/getOffsetTop.js↩
15. https://github.com/cferdinandi/dom-injection-source-code/↩