**JavaScript Basics #6: The Document Object Model**

Starting from this article, we are going to dive into the practical application of the JavaScript language in web development. We’ll talk about how JavaScript, HTML and CSS can work together to make your web pages more appealing and interactive.

Table of Contents

* [The DOM Structure](https://www.ericsdevblog.com/index.php/2021/10/javascript-basics-6-the-document-object-model/#The_DOM_Structure)
* [JavaScript and HTML](https://www.ericsdevblog.com/index.php/2021/10/javascript-basics-6-the-document-object-model/#JavaScript_and_HTML)
  + [Finding Elements](https://www.ericsdevblog.com/index.php/2021/10/javascript-basics-6-the-document-object-model/#Finding_Elements)
  + [Adding and Deleting Elements](https://www.ericsdevblog.com/index.php/2021/10/javascript-basics-6-the-document-object-model/#Adding_and_Deleting_Elements)

**THE DOM STRUCTURE**

Let’s start with a quick review of the document object model. Here is a simple HTML document:

<!DOCTYPE html>

<html lang="en">

<head>

<title>Example Page</title>

</head>

<body>

<h1>Example Page</h1>

<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.Lorem ipsum dolor sit amet, consectetur adipiscing elit.</p>

<div>

<p>Vestibulum fringilla lorem ac elit suscipit, nec suscipit nunc posuere.</p>

<p>Proin efficitur eros scelerisque consequat <a href="https://www.ericsdevblog.com/">pulvinar</a>.</p>

</div>

</body>

</html>

Each HTML element can be seen as a box. For example, the example document has the following structure:

Diagram

Description automatically generated with low confidence

For each box, there is a corresponding object, which we can interact with using JavaScript to find out more details about that box, such as its content, attributes etc. This kind of representation is referred to as the document object model, or DOM for short.

The second important feature about this DOM structure is that the boxes are all connected to each other, which means if we pick a starting point, it is possible for us to move to any other node on the page.

For example, the <body> node has three child elements, <h1>, <p> and <div>. The <div> node has another two paragraphs (<p>) has child elements. So, to locate the paragraph with a link (<a>) in this example document, we can go from <html> to <body> to <div> and finally, locate the <p> node.

**JAVASCRIPT AND HTML**

To import JavaScript code into an HTML document, we can use the <script> </script> tag.

<!DOCTYPE html>

<html lang="en">

<head>

<title>Example Page</title>

</head>

<body>

...

<!--JavaScript-->

<script>

...

</script>

<!--JavaScript in External File-->

<script src="myScript.js"></script>

</body>

</html>

It is customary to put the JavaScript code before the end of the <body> tag. There are two ways to insert JavaScript, just like CSS, you can put it together with HTML, or you can put JavaScript in a separate file. For this tutorial, to make things easier, we’ll put HTML and JavaScript code together.

JavaScript treats each DOM box as an object, and that allows us to access any element in the HTML document using the global binding document. For example, document.body refers to the <body> element of the document.

<body>

...

<!--JavaScript-->

<script>

// Access body element

let body\_element = document.body;

// Access h1 element

let h1\_element = document.body.firstElementChild;

console.log(h1\_element.tagName);

// Access paragraph element (with link)

let p\_element = document.body.childNodes[5].lastElementChild;

console.log(p\_element.tagName);

</script>

</body>

Go to Developer Tools -> Console, and you should see that the correct tag names have been returned.

Notice that the index number we use to locate the <div> element is 5, that is because the childNodes() method will return not only element nodes but also text nodes and comment nodes. For example, a paragraph element would have an element node <p>, and a text node, which is its content.

In web development, it is possible to reach any specific element in the document by starting at document.body and following a fixed path of properties. However, even though that’s possible, it’s still a bad idea, especially when you have a big HTML document with a complicated relationship tree. It is very easy to make a mistake. Luckily, JavaScript offers us some smarter ways of locating elements in an HTML document.

**Finding Elements**

We mentioned before that JavaScript treats all HTML elements as objects, which implies there are built-in methods for us to use. In fact, there are three methods that can help us locate elements in an HTML file, and they actually work a lot like the selectors we talked about in our CSS course.

For instance, all HTML elements have a getElementsByTagName() method, which helps us locate elements with a specific tag.

<body>

<h1>Example Page</h1>

<p class="paragraphs">Lorem ipsum dolor sit amet, consectetur adipiscing elit.Lorem ipsum dolor sit amet, consectetur adipiscing elit.

</p>

<div>

<p class="paragraphs paragraphs\_div">Vestibulum fringilla lorem ac elit suscipit, nec suscipit nunc posuere.</p>

<p class="paragraphs paragraphs\_div" id="paragraph\_link">Proin efficitur eros scelerisque consequat <a href="https://www.ericsdevblog.com/">pulvinar</a>.</p>

</div>

<!--JavaScript-->

<script>

// Get all paragraph elements

let p\_elements = document.body.getElementsByTagName("p");

</script>

</body>

This method will return a collection of elements with the specified tag, and you can access each one of them by specifying the index number (like an Array).

<!--JavaScript-->

<script>

// Get all paragraph elements

let p\_elements = document.body.getElementsByTagName("p");

// Get the second paragraph element and print its content

let second\_p = p\_elements[1];

console.log(second\_p.innerHTML);

</script>

However, do not confuse this collection with an actual array, they are very similar, but not entirely the same. We cannot loop over it using for/of loop, we have to use the index numbers and run over the elements using a regular for loop. Or we can transform this collection into an array using Array.from method.

Once we’ve found the element we are looking for, we can access the attribute and content of that element using the dot (.) operator, and we can change their values as well:

<!--JavaScript-->

<script>

// Get all paragraph elements

let p\_elements = document.body.getElementsByTagName("p");

// Get the second paragraph element

let second\_p = p\_elements[1];

// Print its content

console.log(second\_p.innerHTML);

// Change its content

second\_p.innerHTML = "Changed content.";

// Print its attributes

console.log(second\_p.attributes);

// Access one of the attributes

console.log(second\_p.getAttribute("class"));

</script>

The second method is document.getElementById(), it is used to find one single element, instead of returning a collection of elements. Note that this method does **not** exist under every element object, **there is no** document.body.getElementById().

<!--JavaScript-->

<script>

// Get an element based on ID

let element = document.getElementById("paragraph\_link");

console.log(element.innerHTML);

</script>

The third method is the most useful one, it helps us locate elements with the same class name. That is getElementsByClassName(), which searches the entire document to find a collection of elements with the specified class name.

**Adding and Deleting Elements**

Finally, it’s time to talk about how to manipulate these HTML elements once we’ve located them. In fact, almost everything in the DOM structure can be changed.

For instance, we can remove an element like this:

// Get an element based on ID, and then remove it from the page

let element = document.getElementById("paragraph\_link");

element.remove();

Or we can create a new element, and add it to the DOM structure:

// Create new paragraph element

let new\_p = document.createElement("p");

// Create content for the new p element

let new\_text = document.createTextNode("This is a new paragraph.");

// Append the content to the p element node

new\_p.appendChild(new\_text);

// Add the new paragraph to the DOM structure

let element = document.getElementById("paragraph\_link");

element.append(new\_p);

Like we mentioned before, a paragraph element should have a <p> element node, followed by a text node representing its content.

We can also replace one element with another:

// Replace a paragraph with the new paragraph

let element = document.getElementById("paragraph\_link");

element.replaceWith(new\_p);

In this article, we briefly talked about how to locate and manipulate HTML elements using JavaScript. However, you may have noticed that all the changes are made instantly when we refresh our browser, which is not very interactive. In the next article, we are going to discuss what other events we can use to trigger JavaScript to perform an action.