COURSEWARE

DOM Manipulation

Contents

Overview

The **Document Object Model (DOM)** is a programming interface for HTML and XML documents. It represents the page so that programs can change the document structure, style, and content.

The DOM represents the document as nodes and objects. That way, programming languages can connect to the page.

A Web page is a document. This document can either be displayed in the browser window or as the HTML source, but it is the same document in both cases. The DOM represents that same document so it can be manipulated / modified with a scripting language, such as JavaScript.

Tutorial

Given that every element in a document - the document as a whole, the head, tables within the document, text within table cells - is apart of the DOM for that document, they can all be access and manipulated using the DOM and JavaScript. Lets have a look at how we might do this.

Accessing the DOM

When you create a script, whether it's inline in a `<script>` element, or included in the Web page by means of a script loading instruction, you can immediately begin using the API for the `document` or `window` elements to manipulate the document itself or to get at the children of that document, which are the various elements in the Web page.

Your DOM programming may be something as simple as the following, which displays an alert message by using the `alert()` function from the `window` object:

```
<body onload="window.alert("Welcome to my home page!");">
```

Alternatively, it may use more sophisticated DOM methods to actually create new content.

Basic JavaScript Methods

Below are a few methods that aid DOM Manipulation. You'd likely use these methods frequently in your code.

1. `querySelector()`

The `querySelector()` method **returns the first element that matches one or more CSS selector**. If no match is found, it returns `null`.

```
let ele = document.querySelector(selector);
```

- `ele` - First matching element, or `null` (if no element matches the selectors)
- `selector` - one or more CSS selector, such as `'#fooId'`, `'.fooClassName'`, `'.class1.class2'`, or `'.class1, .class2'`

Before `querySelector()` was introduced, developers widely used the `getElementById()` method, which fetches an element with a specified ID value. Although it is still a useful method, with the newer `querySelector()` and `querySelectorAll()` we are free to target elements based on **any** CSS selector.

In the below example, the first `<div>` gets selected with the `querySelector()` method and its colour is changed to red:

```
<p> Paragraph One<p>
<p> Paragraph Two<p>
<div> Div One </div>
<p> Paragraph Three </p>
<div> Div Two </div>
```

```
let firstDiv = document.querySelector('div');
firstDiv.style.color = 'red';
```

2. `querySelectorAll()`

Unlike `querySelector()`, which returns only the first instance of all matching elements, `querySelectorAll()` **returns all elements that match the specified selector**.

The matching elements are returned as a `NodeList` object, that will be an empty object, if no matching elements are found:

```
let eles = document.querySelectorAll(selector);
```

The example below uses the same HTML as the previous one. However, in this example, all paragraphs are selected with `querySelectorAll()` and are all coloured blue:

```
// HTML same as before

let paras = document.querySelectorAll('p');
for(let p of paras){
    p.style.colour = 'blue';
}
```

3. `addEventListener()`

Events refer to what happens to HTML elements (e.g. clicking, focusing, loading).

We can react to these events with JavaScript by ssigning JS functions to *listen* for these events in elements and do something when the event has occurred.

There are multiple ways you can assign a function to a certain event:

If `foo()` is the custom function, you can register it as a **click event listener** (call it when the button element is clicked):

Method 1 - HTML:

```
<button onclick=foo>Alert</button>
```

Method 2 - JavaScript (1):

```
let btn = document.querySelector('button');
btn.onclick=foo;
```

Method 3 - JavaScript (2):

```
let btn = document.querySelector('button');
btn.addEventListener('click',foo);
```

The final method has some pros: it is the latest standard - allowing the assignment of more than one function as event listeners to one event - and come with a full set of options.

For more information on Event Listeners, refer to the Handling Events module.

4. `createElement()`

The `createElement` method **creates a new HTML element** using the *name of the HTML tag* to be created such as `<p>` or `<div>`.

```
document.createElement(tagName);
```

With the following example we can create a new paragraph element:

```
let pEle = document.createElement('p');
```

5. `appendChild()`

We can add the above element to a Web page by using methods for DOM insertion such as `appendChild()`.0

The `appendChild()` method **adds an element as the last child** to the HTML element that invokes this method.

The child to be inserted can be either a *newly created element* or, an already *existing element*. In the latter case, it will be moved from its previous position to the position of the last child:

```
ele.appendChild(childEle);
```

- `ele` - The HTML element to which we wish to add a child to
- `chileEle` - The HTML element added as the last child of `ele`

The below example inserts a `<strong>` element as the child of a `<div>` element using `appendChild()` and `createElement()`:

```
<div></div>
```

```
let div = document.querySelector('div');
let strong = document.createElement('strong');
strong.textContent = "Hello friends";
div.appendChild(strong);
```

6. `removeChild()`

The `removeChild()` method **removes a specified child element** from the HTML element that calls this method:

```
ele.removeChild(childEle);
```

In this example, we can remove the `<strong>` element we added as a child to the `<div>` tag from the previous `appendChild()` method:

```
div.removeChild(strong);
```

7. `replaceChild()`

The `replaceChild()` method replaces a child element with another one belonging to the parent element that calls this method:

```
ele.replaceChild(newChildEle, oldChildEle);
```

- `ele` - Parent element of which children are to be replace.
- `newChildEle` - Child element of `ele` that will replace `oldChildEle`.
- `oldChildEle` - Child element of `ele` that will be replaced by `newChildEle`.

In the example below, the child element `<strong>` belonging to `<div>` is replaced with the newly created `<em>` tag:

```
<div>
    <strong>hello</strong>
<div>
```

```
let em = document.createElement('em');
let strong = document.querySelector('strong');
let div = document.querySelector('div');
em.textContent = "Replaced!";
div.replaceChild(em,strong);
```

8. `setAttribute()`

The `setAttribute()` method either **adds a new attribute** to an HTML element, or **updates the value** of an attribute that already exists:

```
ele.setAttribute(name,value);
```

In the below example, we add the `contenteditable`' attribute to a `<div>` by making use of the `setAttribute()` method, which will turn its content editable:

```
<div>hello</div>
```

```
let div = document.querySelector('div');
div.setAttribute('contenteditable','');
```

9. `getAttribute()`

The `getAttribute()` method returns the value of a specified attribute belonging to a certain HTML element:

```
ele.getAttribute(name);
```

In the below example, we alert the value of the `contenteditable` attribute belonging to the `<div>` element with the help of the `getAttribute()` method:

```
<div contenteditable=true>hello</div>
```

```
let div = document.querySelector('div');
alert(div.getAttribute('contenteditable'));
```

10. `removeAttribute()`

The `removeAttribute()` method removes a given attribute of a specific HTML element:

```
ele.removeAttribute(name);
```

In this example, we remove the `contenteditable` attribute belonging to the `<div>` element. As a result, the `<div>` becomes uneditable:

```
<div contenteditable=true>hello</div>
```

```
let div = document.querySelector('div');
div.removeAttribute('contenteditable');
```

These are just a few of the methods that we can use to access and manipulate the DOM. For some more information, have a look at the Mozilla DOM introduction.

Exercises

1. Write a function that creates a new `h1` element, adds text to that element and then adds the `h1` to the tree of the document on load of the HTML page.

   ▶ Solution
2. Using the methods taught in this module create a HTML document that applies the basic JavaScript methods to access and manipulate the DOM.