Session 20

Mohamed Emary

June 3, 2024

1 DOM (Document Object Model)

When the browser loads an HTML document, it creates a tree-like structure in memory. This structure is called the Document Object Model (DOM). The DOM represents the document as nodes and objects, allowing you to interact with the document using JavaScript.

For example if you have an `<img />` tag in your HTML, this tag will be represented as an object in the DOM, and its attributes (e.g., `src`, `alt`, `width`, `height`) will become properties of this object and can be accessed and modified using JavaScript.

But what is the difference between HTML and DOM? In short, HTML represents the *initial page content* and the DOM (Document Object Model) represents the *current content* in a tree of objects. If you have a html page and add a tag with javascript, the actual HTML of the page is still the same, but the "DOM" however has changed.

2 Selecting Elements in the DOM

Suppose you have this HTML element:

```
<div id="myElement" class="myClass"></div>
```

To select that element using JavaScript there are several ways to do that. Here are some common methods:

1. `getElementById`: This method returns the element with the specified ID. (Note: IDs must be unique within the document.)

```
var element = document.getElementById('myElement');
```

2. `getElementsByClassName`: This method returns a *collection* of elements with the specified class name.

```
var elements = document.getElementsByClassName('myClass');
```

3. `getElementsByTagName`: This method returns a *collection* of elements with the specified tag name.

```
var elements = document.getElementsByTagName('div');
```

4. `getElementsByName`: This method returns a *node list* of elements with the specified name attribute.

```
var elements = document.getElementsByName('myName');
```

5. `querySelector`: This method returns *the first element* that matches the specified CSS selector.

```
var element = document.querySelector('.myClass');
```

6. `querySelectorAll`: This method returns a *node list* of elements that match the specified CSS selector.

```
var elements = document.querySelectorAll('.myClass');
```

`getElementsByClassName`, `getElementsByTagName` return a *collection* of elements called `HTMLCollection`, which is an *array-like object*. If you want to access a specific element, you can use the index like `elements[0]`.

Since `HTMLCollection` is not an actual array, you can't use array methods like `join`, `push`, `pop`, etc. To convert it to an array, you can use the `Array.from` method.

```
var elements = document.getElementsByClassName ('myClass');
var elementsArray = Array.from(elements);
```

Now you can use array methods on `elementsArray`.

`getElementsByName`, `querySelectorAll` return a `NodeList`, which is also an array-like object that you can loop through and access elements by index and you can't use array methods on it. You can also convert it to an array using `Array.from`.

The difference between `HTMLCollection` and `NodeList` is that `NodeList` is a list of nodes, not just elements. For example, it can contain text nodes, comment nodes, etc. While `HTMLCollection` only contains elements.

There is some elements that are built-in properties of the `document` object:

1. `document.documentElement`: Returns the `<html>` element.
2. `document.head`: Returns the `<head>` element.
3. `document.body`: Returns the `<body>` element.
4. `document.title`: Returns the title of the document.
5. `document.images`: Returns a collection of all `<img>` elements in the document.
6. `document.links`: Returns a collection of all `<a>` elements with a `href` attribute in the document.
7. `document.forms`: Returns a collection of all `<form>` elements in the document.
8. `document.scripts`: Returns a collection of all `<script>` elements in the document.
9. `document.styleSheets`: Returns a collection of all `<link>` and `<style>` elements that have a `rel` attribute with the value `stylesheet`.

3 Event Listeners

Event listeners are used to listen for events on a specific element and execute a JavaScript function when that event occurs. You can add event listeners to any element in the DOM.

Syntax:

```
element.addEventListener(event, function);
```

- `element`: The element to attach the event listener to.
- `event`: The event to listen for (e.g., `click`, `mouseover`, `keydown`, etc.).
- `function`: The function to execute when the event occurs.

This way of adding event listeners is better than using the `onEvent` attribute in the HTML because it allows you to add multiple event listeners to the same element and separate the JavaScript code from the HTML.

Here is an example of adding an event listener to a button element:

In HTML:

```
<button id="myButton">Click me</button>
```

In JavaScript:

```
function sayHello() {
  console.log('Hello!');
}

var button = document.getElementById('myButton');
// Don't use () after function name
button.addEventListener('click', sayHello);
```

Using `()` after the function name will execute the function immediately once the event listener is added. You should only pass the function name without `()`.

But what if that function has parameters? You can use an anonymous function to pass the parameters:

```
function sayHello(name) {
  console.log('Hello, ' + name + '!');
}

button.addEventListener('click', function() {
  sayHello('John');
});
```

The same applies to `element.event` way of adding event listeners:

```
function sayHello() {
  console.log('Hello!');
}
button.onclick = sayHello;
```

And if the function has parameters:

```
function sayHello(name) {
  console.log('Hello, ' + name + '!');
}
button.onclick = function() {
  sayHello('John');
};
```

What is the difference between `addEventListener` and `element.event`? The main difference is that `addEventListener` allows you to add multiple event listeners to the same element, while `element.event` can only have one event listener per event type.

```
var button = document.getElementById('myButton');
button.onclick = function() {
  console.log('Hello!');
};

button.onclick = function() {
  console.log('Goodbye!');
};
```

In this example, only the second event listener will be executed because the first one will be overwritten.

```
var button = document.getElementById('myButton');
button.addEventListener('click', function() {
  console.log('Hello!');
});

button.addEventListener('click', function() {
  console.log('Goodbye!');
});
```

In this example, both event listeners will be executed in the order they were added.

4 Event Object

When an event occurs, the browser creates an event object that contains information about the event. This object is passed as an argument to the event listener function.

Here is an example of using the event object to get information about a click event:

```
var button = document.getElementById('myButton');
button.addEventListener('click', function(event) {
  console.log(event);
});
```

The event object contains information such as:

- `type`: The type of event (e.g., `click`, `dblclick`, `mouseover`, `keydown`, etc.).
- `target`: The element that triggered the event.
- `clientX`, `clientY`: The coordinates of the mouse pointer when the event occurred.
- `keyCode`: The key code of the key that was pressed (for keyboard events).

Example using the event object with the whole document:

```
document.addEventListener('keydown', function(event) {
  console.log(event.keyCode);
});

document.addEventListener('click', function(event) {
  console.log(event.clientX, event.clientY);
  console.log(event.target);
  console.log(event.type);
});
```

5 Some Common Events

Here are some common events that you can listen for:

Mouse Related Events:

- `click`: The user clicks an element.
- `dblclick`: The user double-clicks an element.
- `mousemove`: The user moves the mouse.
- `mouseenter`: The user moves the mouse over an element.
- `mouseleave`: The user moves the mouse out of an element.
- `mouseup`: The user releases a mouse button.
- `mousedown`: The user presses a mouse button.
- `mouseover`: The user moves the mouse over an element.
- `mouseout`: The user moves the mouse out of an element.
- `scroll`: The user scrolls the page.
- `drag`: The user is dragging an element. (Note: you should add `draggable="true"` to the element HTML code to make it draggable.)
- `dragstart`: The user starts dragging an element.
- `dragend`: The user stops dragging an element.

Keyboard Related Events:

- `keyup`: The user releases a key on the keyboard.
- `keydown`: The user presses a key on the keyboard.
- `keypress`: The user presses a key on the keyboard.

Input and Form Related Events:

- `input`: The user inputs text into an input element.
- `change`: The user focuses out of an input element after changing its value.
- `submit`: The user submits a form. This for example can be used to prevent the page from reloading when submitting a form using `event.preventDefault()` (`event` here is the event object passed to the event listener function not the event type).

Focus Related Events:

- `focus`: The user focuses on an input element.
- `blur`: The user focuses out of an input element.

There are many more events that you can listen for. You can find a complete list of events in the MDN Web Docs.

6 Changing Element Styles

You can change the style of an element using JavaScript by accessing its `style` property. This property contains all the CSS properties of the element.

Here is an example of changing the background color of a `div` element:

```
var element = document.getElementById('myElement');
element.style.backgroundColor = 'red';
```

You can also change multiple styles at once using the `cssText` property:

```
element.style.cssText = 'background-color: red; color: white; font-size:
↪  20px;';
```

These properties are useful for changing styles dynamically based on user interactions or other events.

The styles applied using the `style` property are inline styles, which have the highest specificity and override any other styles defined in external CSS files or internal styles except ones with `!important`.

When a style has `!important` and you want to override it using JavaScript, you can use the `cssText` property with `!important`:

```
element.style.cssText = 'background-color: red !important;';
```

6.1 Example of Making an Element Draggable

To make an element draggable, you need to add the `draggable` attribute to the element and set it to `true`. You can then listen for the `dragend` event to get the mouse coordinates and move the element to that position.

Here is an example of making a `div` element draggable:

In HTML:

```
<div id="myElement" draggable="true">Drag me</div>
```

In JavaScript:

```
var element = document.getElementById("myElement");
element.style.cssText = `
  width: 100px;
  height: 100px;
  line-height: 100px;
  text-align: center;
  background-color: gold;`;
element.addEventListener("dragend", function (event) {
```

```
  element.style.position = "absolute";
  element.style.left = event.clientX + "px";
  element.style.top = event.clientY + "px";
  element.style.transform = "translate(-50%, -50%)";
});
```

In this code we applied some styles to the element using the `cssText` property, then we listened for the `dragend` event to get the mouse coordinates and move the element to that position.

Don't forget to add `px` after because the `clientX` and `clientY` properties return the mouse coordinates in pixels but the unit is not specified so you need to add `px` after the value.

In the example above you can also use `mousemove` event instead of `dragend` to move the element while dragging it but that will make you not able to drop it in a new position. You can for example use that to make a simple icon that is always following the mouse cursor or to make a simple drawing app where you draw by dragging the mouse.

7 Get, Set, and Remove Attributes

You can use the `setAttribute` method to set an attribute of an element and the `getAttribute` method to get the value of an attribute.

Here is an example of setting and getting the `src`, `alt` attributes of an `img` element:

```
var img = document.getElementById('myImage');
img.setAttribute('src', 'image.jpg');
img.setAttribute('alt', 'My Image');

var src = img.getAttribute('src');
var alt = img.getAttribute('alt');
console.log(src); // Output: image.jpg
console.log(alt); // Output: My Image
```

You can also use these methods to set and get styles:

```
<div id="myElement" style="background-color: red;">lorem</div>
```

```
var element = document.getElementById('myElement');
var backgroundColor = element.getAttribute('style');
console.log(backgroundColor); // Output: background-color: red;

element.setAttribute('style', 'background-color: blue;');
backgroundColor = element.getAttribute('style');
console.log(backgroundColor); // Output: background-color: blue;
```

To remove an attribute, you can use the `removeAttribute` method:

```
element.removeAttribute('style');
```

This will remove the `style` attribute from the element.