Session 22

Mohamed Emary

June 10, 2024

1 `innerHTML` and `innerText`

1.1 `innerHTML`

- `innerHTML` returns the HTML content of an element.
- When assigning a value with HTML tags to `innerHTML`, the browser will render the HTML tags as HTML elements.
- When printing the value of `innerHTML` of an element that contains HTML tags, the browser will show the HTML tags in the output.

Example:

HTML:

```
1  <p id="example">My <strong>example</strong> paragraph</p>
```

JavaScript:

```
1  var example = document.getElementById('example');
2  console.log(example.innerHTML); // My <strong>example</strong> paragraph
3
4  example.innerHTML = 'My <strong>new</strong> paragraph';
5  console.log(example.innerHTML); // My <strong>new</strong> paragraph
```

1.2 `innerText`

- `innerText` returns the text content of an element.
- When assigning a value with HTML tags to `innerText`, the browser will render the HTML tags as plain text.
- When printing the value of `innerText` of an element that contains HTML tags, the browser will show the HTML tags in the output.

Example:

HTML:

```
1  <p id="example">My <strong>example</strong> paragraph</p>
```

JavaScript:

```
1  var example = document.getElementById('example');
2  console.log(example.innerText); // My example paragraph
3
4  example.innerText = 'My <strong>new</strong> paragraph';
5  console.log(example.innerText); // My <strong>new</strong> paragraph
```

Here the `<strong>new</strong>` appear in the web page as it is because `innerText` does not render HTML tags.

2 Creating Elements

To create an element, you can use the `document.createElement()` method. This method creates a new element with the specified tag name.

Example:

```
1  var newElement = document.createElement('div');
```

To set the attributes of the new element, you can use the `setAttribute()` method or the . notation.

Example:

```
1  // Using the setAttribute() method
2  newElement.setAttribute('id', 'new-element');
3  newElement.setAttribute('class', 'new-class');
4
5  // Or using the . notation
6  newElement.id = 'new-element';
7  newElement.className = 'new-class';
```

2.1 Appending & Prepending Elements (Child)

To append an element inside another element in the DOM, you can use the `append()` method, and to prepend an element, you can use the `prepend()` method.
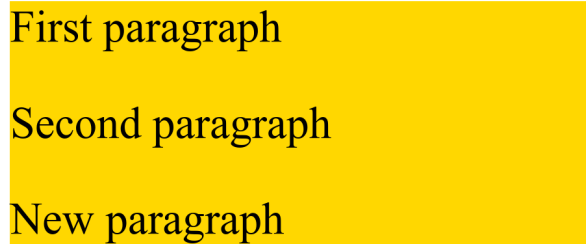
Example:

HTML:

```
1  <div id="parent" style="background-color: gold">
2    <p>First paragraph</p>
3    <p>Second paragraph</p>
4  </div>
```

JavaScript:

```
1  var parent = document.getElementById('parent');
2  var newElement = document.createElement('p');
3  newElement.innerText = 'New paragraph';
4
```

```
5  // Append the new element inside the parent element
6  parent.append(newElement);
```

Now the result will look like this:

First paragraph

Second paragraph

New paragraph

Figure 1: Using `append`

2.2   Add Element Before or After Another (Sibling)

To add an element before or after another element in the DOM, you can use the `before()` and `after()` methods.

Example:
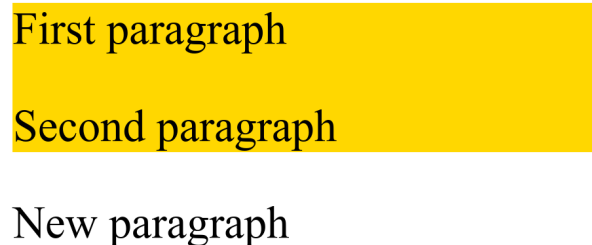
HTML:

```
1  <div id="parent" style="background-color: gold">
2    <p>First paragraph</p>
3    <p>Second paragraph</p>
4  </div>
```

JavaScript:

```
1  var parent = document.getElementById('parent');
2  var newElement = document.createElement('p');
3  newElement.innerText = 'New paragraph';
4
5  // Add the new element after the second paragraph
6  parent.after(newElement);
```

Now the result will look like this:

First paragraph

Second paragraph

New paragraph

Figure 2: Using `after`

> **Note**
>
> You can only send elements as arguments to the `append()`, `prepend()`, `before()`, and `after()` methods. If you send HTML tag or text, it will be treated as a string and not as an element.

3   Traversing the DOM

Traversing the DOM which is a way to move around the DOM tree and select elements based on their relationship to other elements.

Some useful properties and methods for traversing the DOM are:

1. `parentElement`: returns the parent **element** of an element.

2. `parentNode`: returns the parent **node** of an element.

3. `firstElementChild`: returns the first child **element** of an element.

4. `lastElementChild`: returns the last child **element** of an element.

5. `children`: returns an **HTML collection** of an element's child elements.

6. `childNodes`: returns a **NodeList** of an element's child nodes.

7. `nextElementSibling`: returns the next sibling **element** of an element.

8. `previousElementSibling`: returns the previous sibling **element** of an element.

9. `nextSibling`: returns the next sibling **node** of an element.

10. `previousSibling`: returns the previous sibling **node** of an element.

Example:

HTML:

```
1  <div id="parent" style="background-color: gold">
2    <p id="p1">First paragraph</p>
3    <p>Second paragraph</p>
4  </div>
```

JavaScript:

```
1  var parent = document.getElementById('parent');
2  var p1 = document.getElementById('p1');
3
4  // Get the parent element of the first paragraph
5  var parentElement = p1.parentElement;
6  console.log(parentElement.id); // parent
7
8  // Get the parent node of the first paragraph
9  var parentNode = p1.parentNode;
10 console.log(parentNode.id); // parent
11
12 // Get the first child of the parent element
13 var firstChild = parent.firstElementChild;
14 console.log(firstChild.innerText); // First paragraph
15
16 // Get the last child of the parent element
17 var lastChild = parent.lastElementChild;
18 console.log(lastChild.innerText); // Second paragraph
19
20 // Get all the child elements of the parent element
21 var children = parent.children;
```

```
22  console.log(children.length); // 2
23  console.log(children[1]); // <p>Second paragraph</p>
24
25  // Get all the child nodes of the parent element
26  var childNodes = parent.childNodes;
27  console.log(childNodes.length); // 3
28  console.log(childNodes[1]); // #text
29
30  // Get the next sibling element of the first paragraph
31  var nextSibling = p1.nextElementSibling;
32  console.log(nextSibling.innerText); // Second paragraph
33
34  // Get the previous sibling element of the second paragraph
35  var previousSibling = lastChild.previousElementSibling;
36  console.log(previousSibling.innerText); // First paragraph
37
38  // Get the next sibling node of the first paragraph
39  var nextNode = p1.nextSibling;
40  console.log(nextNode); // #text
41
42  // Get the previous sibling node of the second paragraph
43  var previousNode = lastChild.previousSibling;
44  console.log(previousNode); // #text
```

4 Important Differences

4.1 `previousSibling`, `nextSibling` VS `previousElementSibling`, `nextElementSibling`

- `previousSibling` and `nextSibling` return nodes and these nodes include both element nodes and non-element nodes (like text and comment nodes).

- `previousElementSibling` and `nextElementSibling` return only element nodes and ignore text and comment nodes.

```
1  <div>
2    <p>Paragraph 1</p>
3    Text
4    <p>Paragraph 2</p>
5  </div>
```

If the current node is the first `<p>` element, `nextSibling` would return the text node `Text`, while `nextElementSibling` would return the second `<p>` element.

4.2 NodeList VS HTML Collection

NodeList	HTML Collection
Static (Does not update when the DOM changes)	Live (Updates when the DOM changes)
Returns a list of nodes	Returns a list of elements

NodeList	HTML Collection
Nodes can be of any type like element, text, comment, etc.	Elements only
Returned by methods like `querySelectorAll`, `childNodes`	Returned by methods like `getElementsByTagName`, `children`

By saying that the `NodeList` is static, it means that if you add an element to the DOM after getting the `NodeList`, the `NodeList` will not include the new element. On the other hand, the `HTML Collection` is live, which means that it will include the new element even after getting the `HTML Collection`.

Example:

HTML:

```
1  <div>
2    <p>Paragraph 1</p>
3    <p>Paragraph 2</p>
4  </div>
```

JavaScript:

```
1   var div = document.querySelector('div');
2   var paragraphsCollection = div.getElementsByTagName('p');
3   var paragraphsNodeList = div.querySelectorAll('p');
4
5   console.log(paragraphsCollection.length); // 2
6   console.log(paragraphsNodeList.length);   // 2
7
8   var newParagraph = document.createElement('p');
9   newParagraph.innerText = 'New paragraph';
10
11  div.append(newParagraph);
12
13  console.log(paragraphsCollection.length); // 3
14  console.log(paragraphsNodeList.length);   // 2
```

In this example, the `paragraphsCollection` will have a length of 3, while the `paragraphsNodeList` will have a length of 2 because the `NodeList` is static and does not change when the DOM changes while the `HTML Collection` is live and changes when the DOM changes.

5   Browser Object Model (BOM)

Browser Object Model or BOM is a set of objects provided by the browser to interact with the browser itself.

5.1   DOM VS BOM

The DOM can be accessed via the BOM through the `window.document` property. So, you can say that the DOM is part of the BOM in a browser environment.

5 BROWSER OBJECT MODEL (BOM)

`window` is a super global object in the browser environment.

DOM is concerned with the content of the web document, while the BOM is concerned with the browser environment.

5.2 BOM Methods

Some of the BOM methods include:

- `setInterval()`: Calls a function or evaluates an expression each time a specified number of milliseconds elapses.

For example, to display the value of a counter every second:

```
function incrementCounter() {
  console.log(counter);
  counter++;
}

var counter = 0;
var interval = setInterval(incrementCounter, 1000);
```

- `clearInterval()`: Stops the intervals set by `setInterval()`.

For example, to stop the counter we made earlier when the user clicks a button:

```
var button = document.getElementById('stop');

button.addEventListener('click', function() {
  console.log("Counter stopped");
  clearInterval(interval);
});
```

- `setTimeout()`: Calls a function or evaluates an expression **once** after a specified number of milliseconds.

For example, to display a message after 3 seconds:

```
function showMessage() {
  console.log("Hello, world!");
}

setTimeout(showMessage, 3000);
```

- `alert()`: Displays an alert box with a message and an OK button.

For example:

```
alert("Hello, world!");
```

- `window.open()`: Opens a new browser window or a new tab.

For example, This will open a new tab with Google's homepage.

```
var googleBtn = document.getElementById('open');
googleBtn.addEventListener('click', function() {
  window.open('https://www.google.com', '_blank');
});
```

`_blank` is the name of the target window. It specifies that the URL should be opened in a new tab and it's the default value. To open the URL in the same tab, you can use `_self`.