Module 4

Handling DOM Events

Creating HTML Dynamically

Updating HTML Content Dynamically

Managing Web Forms with JavaScript

Element: innerHTML property

The `Element` property `innerHTML` gets or sets the HTML or XML markup contained within the element.

Value

A string containing the HTML serialization of the element's descendants. Setting the value of `innerHTML` removes all of the element's descendants and replaces them with nodes constructed by parsing the HTML given in the string htmlString.

Usage notes

The `innerHTML` property can be used to examine the current HTML source of the page, including any changes that have been made since the page was initially loaded.

Reading the HTML contents of an element

Reading `innerHTML` causes the user agent to serialize the HTML or XML fragment comprised of the element's descendants. The resulting string is returned.

```
let contents = myElement.innerHTML;
```

This lets you look at the HTML markup of the element's content nodes.

> Note: The returned HTML or XML fragment is generated based on the current contents of the element, so the markup and formatting of the returned fragment is likely not to match the original page markup.

Codepen

Replacing the contents of an element

Setting the value of innerHTML lets you easily replace the existing contents of an element with new content.

> Note: This is a security risk if the string to be inserted might contain potentially malicious content. When inserting user-supplied data you should always consider using `Element.setHTML()` instead, in order to sanitize the content before it is inserted

For example, you can erase the entire contents of a document by clearing the contents of the document's body attribute:

```
document.body.innerHTML = "";
```

Example: Adding contents to an element

HTML:

```html
<ul id="list">
  <li><a href="#">Item 1</a></li>
  <li><a href="#">Item 2</a></li>
  <li><a href="#">Item 3</a></li>
</ul>
```

JS:

```js
const list = document.getElementById("list");

list.innerHTML += `<li><a href="#">Item ${list.children.length + 1}</a></li>`;
```

> Please note that using innerHTML to append HTML elements (e.g. el.innerHTML += "link") will result in the removal of any previously set event listeners. That is, after you append any HTML element that way you won't be able to listen to the previously set event listeners.

Codepen

HTMLElement: innerText property

The innerText property of the `HTMLElement` interface represents the rendered text content of a node and its descendants.

As a getter, it approximates the text the user would get if they highlighted the contents of the element with the cursor and then copied it to the clipboard. As a setter this will replace the element's children with the given value, converting any line breaks into `<br>` elements.

Codepen

Document: createElement() method

In an HTML document, the `document.createElement()` method creates the HTML element specified by *tagName*, or an `HTMLUnknownElement` if *tagName* isn't recognized.

Syntax:

```
createElement(tagName)
```

`tagName`

A string that specifies the type of element to be created. The `nodeName` of the created element is initialized with the value of *tagName*. Don't use qualified names (like "html:a") with this method. When called on an HTML document, `createElement()` converts tagName to lower case before creating the element.

Document: createTextNode() method

Creates a new `Text` node. This method can be used to escape HTML characters.

Syntax

```
createTextNode(data)
```

`data`

A string containing the data to be put in the text node.

Node: appendChild() method

The `appendChild()` method of the `Node` interface adds a node to the end of the list of children of a specified parent node.

Syntax

```
appendChild(aChild)
```

`aChild`

The node to append to the given parent node (commonly an element).

Node: insertBefore() method

The `insertBefore()` method of the `Node` interface inserts a node before a reference node as a child of a specified parent node.

If the given node already exists in the document, `insertBefore()` moves it from its current position to the new position. (That is, it will automatically be removed from its existing parent before appending it to the specified new parent.)

This means that a node cannot be in two locations of the document simultaneously.

Syntax

```
insertBefore(newNode, referenceNode)
```

`newNode`
The node to be inserted.
`referenceNode`
The node before which newNode is inserted. If this is null, then newNode is inserted at the end of node's child nodes.

Example 1:

HTML:

```html
<!doctype html>
<html lang="en-US">
  <head>
    <meta charset="UTF-8" />
    <title>Working with elements</title>
  </head>
  <body>
    <div id="div1">The text above has been created dynamically.</div>
  </body>
</html>
```

JS:

```
document.body.onload = addElement;

function addElement() {
  // create a new div element
  const newDiv = document.createElement("div");

  // and give it some content
  const newContent = document.createTextNode("Hi there and greetings!");

  // add the text node to the newly created div
  newDiv.appendChild(newContent);

  // add the newly created element and its content into the DOM
  const currentDiv = document.getElementById("div1");
  document.body.insertBefore(newDiv, currentDiv);
}
```

Codepen

Example 2:

```html
<input id="items"><button>Add to shopping list</button>
```

```javascript
const inputElement = document.querySelector("input#items");
const btn = document.querySelector("button");

const ul = document.createElement("ul");
btn.after(ul);

btn.addEventListener("click", () => {
  const item = inputElement.value;
  console.log(item);
  const li = document.createElement("li");
  const txt = document.createTextNode(item);
  li.appendChild(txt);
  ul.appendChild(li);
  inputElement.value = "";
});
```

Codepen

Example 3:

Display Banner Example Demo

DOMParser

The `DOMParser` interface provides the ability to parse XML or HTML source code from a string into a DOM Document.

You can perform the opposite operation—converting a DOM tree into XML or HTML source—using the XMLSerializer interface.

In the case of an HTML document, you can also replace portions of the DOM with new DOM trees built from HTML by setting the value of the Element.innerHTML and outerHTML properties. These properties can also be read to fetch HTML fragments corresponding to the corresponding DOM subtree.

Note that XMLHttpRequest can parse XML and HTML directly from a URL-addressable resource, returning a Document in its response property.

DOMParser: parseFromString() method

The `parseFromString()` method of the `DOMParser` interface parses a string containing either HTML or XML, returning an `HTMLDocument` or an `XMLDocument`.

Syntax

```
parseFromString(string, mimeType)
```

`string`

The string to be parsed. It must contain either an `HTML`, xml, XHTML, or svg document.

`mimeType`

A string. This string determines whether the XML parser or the HTML parser is used to parse the string.

Valid mimetypes

- text/html
- text/xml
- application/xml
- application/xhtml+xml
- image/svg+xml

A value of `text/html` will invoke the HTML parser, and the method will return an `HTMLDocument`. Any `<script>` element gets marked non-executable, and the contents of `<noscript>` are parsed as markup.

The other valid values (`text/xml`, `application/xml`, `application/xhtml+xml`, and `image/svg+xml`) are functionally equivalent. They all invoke the XML parser, and the method will return an `XMLDocument`.

Any other value is invalid and will cause a `TypeError` to be thrown.

Example:

```
const parser = new DOMParser();

const xmlString = "<warning>Beware of the tiger</warning>";
const doc1 = parser.parseFromString(xmlString, "application/xml");
// XMLDocument

const svgString = '<circle cx="50" cy="50" r="50"/>';
const doc2 = parser.parseFromString(svgString, "image/svg+xml");
// XMLDocument

const htmlString = "<strong>Beware of the leopard</strong>";
const doc3 = parser.parseFromString(htmlString, "text/html");
// HTMLDocument

console.log(doc1.documentElement.textContent);
// "Beware of the tiger"

console.log(doc2.firstChild.tagName);
// "circle"

console.log(doc3.body.firstChild.textContent);
// "Beware of the leopard"
```

Codepen

Semi-Practical example:

```
const template = `
  <header>
      <nav>
          <ul>
              <li>Menu item 1</li>
              <li>Menu item 2</li>
          </ul>
      </nav>
      <h1 id="title">Page Title</h1>
  </header>
`;

const parser = new DOMParser();
const parsedDocument = parser.parseFromString(template, "text/html");
const domObjects = parsedDocument.body.firstChild;
console.log(domObjects);

document.body.append(domObjects);

// Add one more menu item
const navUl = document.querySelector("nav ul");
navUl.innerHTML += `<li>Menu item 3</li>`;
```

See dom-parser.html demo

Todos

GitHub Classroom

JS1 Lesson 4.2 Creating HTML Dynamically

Mollify

Read Creating HTML Dynamically, and do the Lesson Task