JavaScript For Web

Week 3, Lecture 3 - JavaScript in Web Development: Document Object Model

Instructor: Jason Xu

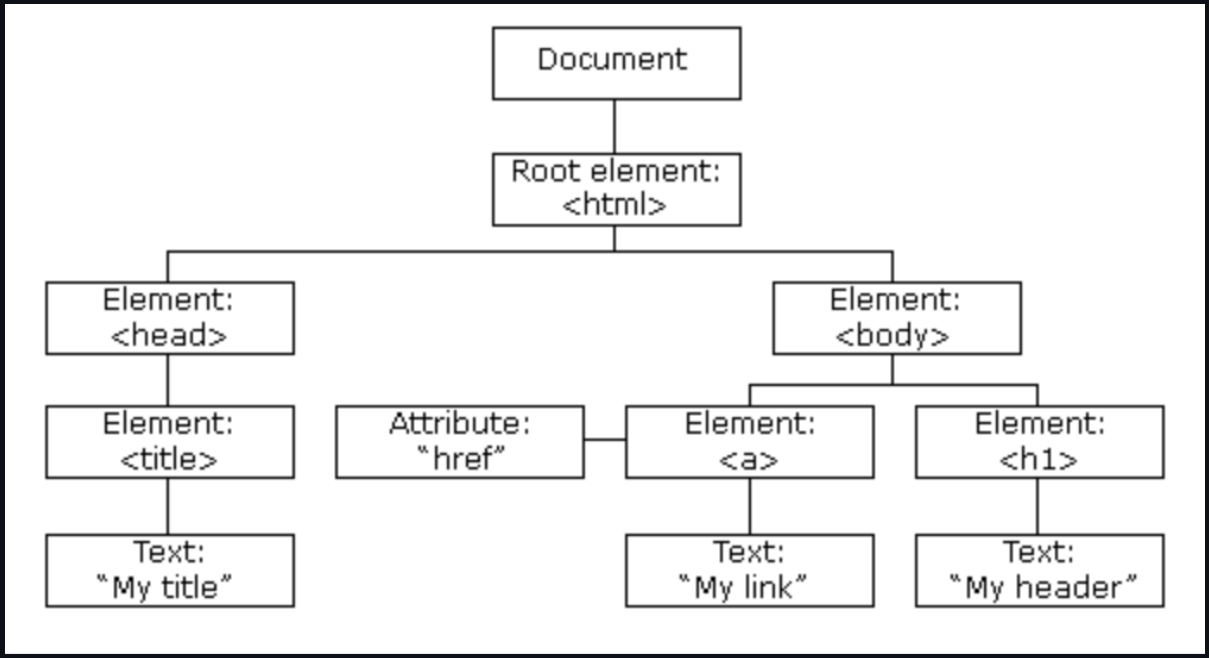
Today's Overview

- What is Document Object Model(DOM)
- DOM Methods
- Tutorial: Readings and reviews

What is Document Object Model (DOM)

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

- Our HTML file, Web page, is a document that can be either displayed in the browser window.
- The DOM represents the document as objects which can interact with the HTML page/content.
- The DOM represents content as a tree structure - a tree of "nodes" with different relationships depending on how they're arranged in the HTML document.



What is Document Object Model (DOM)

The DOM represents content as **a tree structure** - a tree of "nodes" with different relationships depending on how they're arranged in the HTML document.

```
<div id="container">
  <div class="display"></div>
  <div class="controls"></div>
</div>
```

In the above example, the `<div class="display"></div>` is a "child" of `<div id="container"></div>` and a sibling to `<div class="controls"></div>`. Think of it like a "family tree". `<div id="container"></div>` is a "parent", with its children on the next level, each on their own "branch".

When working with the DOM, you use "selectors" to target the nodes you want to work with. You can also use relational selectors (i.e. `firstElementChild` or `lastElementChild` etc.) with special properties owned by the nodes.

```
const container = document.querySelector('#container')
// selects the #container div (don't worry about the syntax, we'll get there)

console.dir(container.firstElementChild)
// selects the first child of #container => .display

const controls = document.querySelector('.controls')
// selects the .controls div

console.dir(controls.previousElementSibling)
// selects the prior sibling => .display
```

DOM Methods

When your HTML code is parsed by a web browser, it is converted to the DOM as was mentioned above. One of the primary differences is that these nodes are objects that have many properties and methods attached to them.

These **properties** and **methods** are the primary tools we are going to use to manipulate our webpage with JavaScript.

- Query selector
- Element creation
- Append elements
- Remove elements
- altering elements

DOM Methods: Query Selector

- `element.querySelector(selector)` - returns a reference to the first match of selector.
- `element.querySelectorAll(selectors)` - return a "nodelist" containing references to all of the matches of the selectors.

For example

```
<div class = "web">
    <h3>Web Dev Notes</h3>
    <ul>
      <li><h3>Keep the browser inspector opened all the time!</h3></li>
      <li><h3>Take small steps</h3></li>
      ...
    </ul>
</div>
```

```
let title = document.querySelector('.web h3')

console.log(title.innerHTML)

let notes = document.querySelectorAll('.web ul > li')

for (let n of notes) {
  console.log(n.textContent)
}
```

DOM Methods: Element Creation

- `document.createElement(tagName)` - creates a new element of tag type tagName.

```
let newNote = document.createElement('li')
newNote.innerHTML = '<h3>this is new notes<h3/>'
```

This function **does NOT** put your new element into the DOM - it simply creates it in memory. This is so that you can manipulate the element (by adding styles, classes, ids, text, etc.) before placing it on the page. You can place the element into the DOM with one of the following methods.

DOM Methods: Append Element

- `parentNode.append(childNode)` - appends childNode as the last child of parentNode.

```
let noteList = document.querySelector('.web ul')

const newNote = document.createElement('li')
newNote.innerHTML = '<h3>this is new notes<h3/>'

noteList.append(newNote)
```

DOM Methods: Remove Element

- `parentNode.removeChild(child)` - removes child from parentNode on the DOM and returns a reference to child.

```
let notes = document.querySelectorAll('.web ul > li')

let lastNote = notes[notes.length - 1]

noteList.removeChild(lastNote)
```

DOM Methods: Altering Element

When you have a reference to an element, you can use that reference to alter the element's own properties. This allows you to do many useful alterations, like adding/removing and altering attributes, changing classes, adding inline style information and more.

```
let title = document.querySelector('.web h3')

title.style.color = 'blue' // set the text color to blue
```

Editing attributes:

```
title.setAttribute('id', 'title-id') // If id exists, update it to 'title-id', else create an id with value 'title id'

title.getAttribute('id') // returns value of specified attribute, in this case "title-id"

title.removeAttribute('id') // removes the specified attribute
```

Working with classes:

```
title.classList.add('new') // adds class "new" to your title

title.classList.remove('new') // removes "new" class from title

title.classList.toggle('active') // if title doesn't have class "active" then add it, or if it does, then remove it
```

Summary

- The DOM is an API represents webpage content as a tree structure object.
- DOM Methods:
  - Query selector: `querySelector()`, `querySelectorAll()`
  - Element creation: `createElement()`
  - Append element: `append()`
  - Remove element: `removeChild()`
  - atering element: `textContent`, `innerHTML`, styles, attributes, classes, etc.

Thank you