Lecture 4.2

Intro to DOM

Topics

- Intro to the DOM
- Document Interface
- Selecting HTML Elements
- Query Selector

The DOM

What is the DOM?

DOM stands for Document Object Model. It's nothing more than the block level diagram of all the HTML elements loaded on the page when a browser loads a web page.

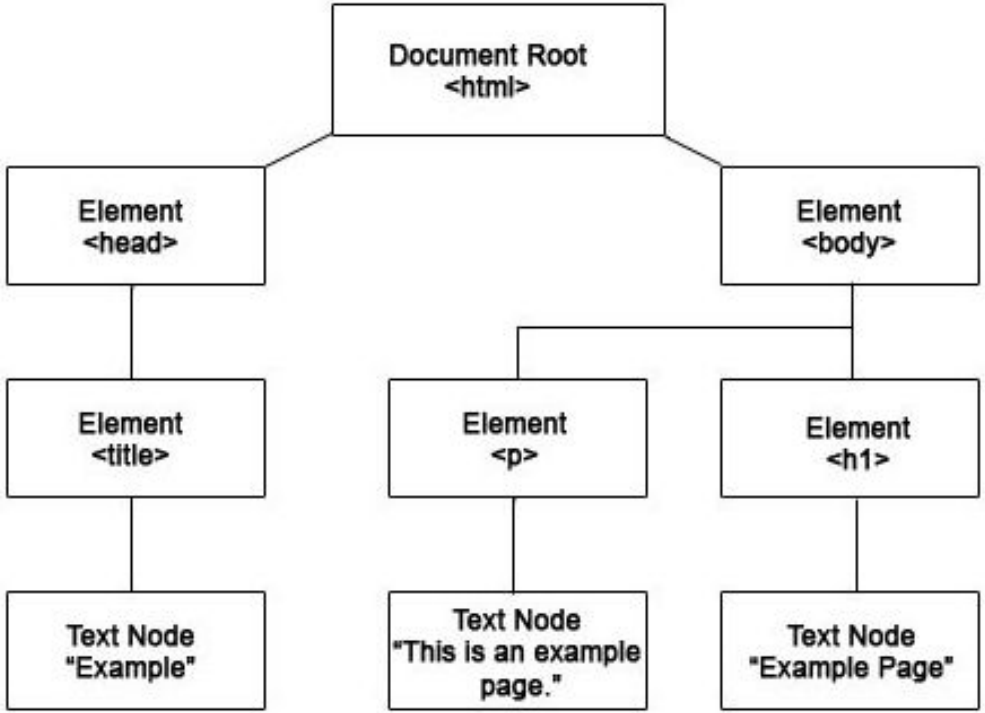
It is presented as a tree of objects which are HTML elements.

DOM allows us to add interactivity to our pages!

The DOM

This document enables Javascript to access and manipulate the elements and styles of a website. The model is built in a tree structure of objects and defines:

- HTML elements as objects
- Properties and events of the HTML elements
- Methods to access the HTML elements



Interacting with the DOM

- The Document Object Model (DOM) is an application programming interface (API).
  - Change/Remove HTML elements in the DOM/on the page
  - Change & add CSS styles to elements
  - Read & change element attributes (href, src, alt, custom)
  - Create new HTML elements and insert them into the DOM/page
  - Attach event listeners to elements (click, keypress, submit)

Interacting with the DOM

- The Document Object Model (DOM) is an application programming interface.
  - Change/Remove HTML elements in the DOM/on the page
  - Change & add CSS styles to elements
  - Read & change element attributes (href, src, alt, custom)
  - Create new HTML elements and insert them into the DOM/page
  - Attach event listeners to elements (click, keypress, submit)

Document Interface

A document as a hierarchy of nodes

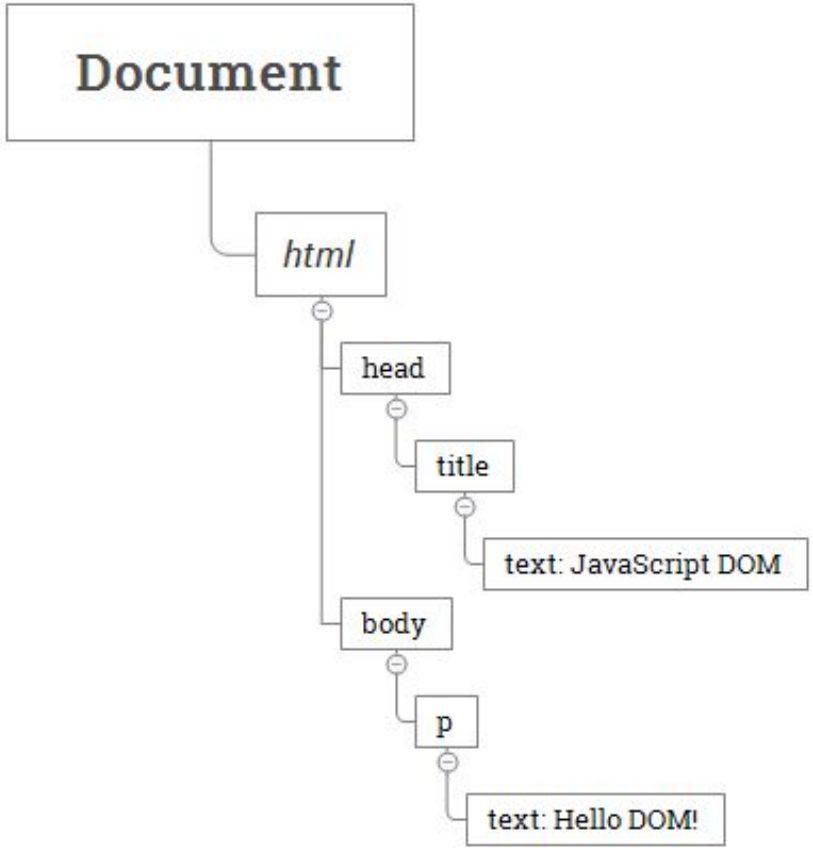
The DOM represents an HTML or XML document as a hierarchy of nodes. Consider the following HTML document:

```
<html>
    <head>
        <title>JavaScript DOM</title>
    </head>
    <body>
        <p>Hello DOM!</p>
    </body>
</html>
```

A document as a hierarchy of nodes

The DOM represents an HTML or XML document as a hierarchy of nodes. Consider the following HTML document:

```
<html>
    <head>
        <title>JavaScript DOM</title>
    </head>
    <body>
        <p>Hello DOM!</p>
    </body>
</html>
```



The following tree represents the above HTML document:

Document API

The **Document** interface represents any web page loaded in the browser and serves as an entry point into the web page's content, which is the DOM tree.

The Document interface describes the common properties and methods for any kind of document. Depending on the document's type (e.g. HTML, XML, SVG, ...), a larger API is available:

HTML documents, served with the "text/html" content type, also implement the HTMLDocument interface, whereas XML and SVG documents implement the XMLDocument interface.

Select HTML Elements

Get element by ID

The *getElementById()* method is used to get a single element by its id.

```
var title = document.getElementById('header-title');
```

Here we get the element with the id of header-title and save it into a variable.

The getElementById() returns a DOM element specified by an id or null if no matching element found.

If multiple elements share the same id, even though it is invalid, the getElementById() returns the first element it encounters.

```
<html>
    <head>
        <title>JavaScript getElementById() Method</title>
    </head>
    <body>
        <p id="message">A paragraph</p>
    </body>
</html>
```

The document contains a <p> element that has the id attribute with the value message:

```
const p = document.getElementById('message');
console.log(p);
```

Output:

```
<p id="message">A paragraph</p>
```

Get elements by class name

We can also get more than one object using the *getElementsByClassName()* method which returns an array of elements.

```
var items = document.getElementsByClassName('list-items');
```

Here we get all items with the class *list-items* and save them into a variable.

The getElementsByName() accepts a *name* which is the value of the *name* attribute of elements and returns a live *NodeList* of elements.

Every element on an HTML document may have a name attribute:

Unlike the id attribute, multiple HTML elements can share the same value of the name attribute like this:

```
<input type="radio" name="language" value="JavaScript">
<input type="radio" name="language" value="TypeScript">
```

To get all elements with a specified name, you use the getElementsByName() method of the document object:

```
let elements = document.getElementsByName(name);
```

Get elements by tag name

We can also get our elements by tag name using the *getElementsByTagName()* method

```
var listItems = document.getElementsByTagName('li');
```

Here we get all *li* elements of our HTML document and save them into a variable.

Get elements by tag name cont..

The getElementsByTagName() is a method of the document object or a specific DOM element.

The getElementsByTagName() method accepts a tag name and returns a live HTMLCollection of elements with the matching tag name in the order which they appear in the document.

```
let elements = document.getElementsByTagName(tagName);
```

Get elements by class name

The getElementsByClassName() method to select elements based on their classes.

```
<button class="btn btn-primary">Save</button>
```

In this syntax, the classNames parameter is a string that represents a class name to match

```
let btn = document.getElementsByClassName('btn');
```

If you match elements by multiple classes, you need to use whitespace to separate them like this:

```
let btn = document.getElementsByClassName('btn bt-primary');
```

Query Selector

querySelector

The *querySelector()* method returns the first element that matches a specified *CSS selector*.

That means that you can get elements by id, class, tag and all other valid CSS selectors. Here I just list a few of the most popular options.

The *querySelectorAll()* method returns the all the elements that matches a specified *CSS selector*.

```
var header = document.querySelector('#header')
```

querySelector - Universal Selector

The universal selector denoted by * that matches all elements of any type:

```
*
```

The following uses querySelector() selects the first element in the document:

```
let element = document.querySelector('*');
```

The following uses querySelectorall() to find all elements in the document:

```
let elements = document.querySelectorAll('*');
```

querySelector - Type Selector

To select elements by node name, you use the type selector e.g., a selects all <a> elements:

```
elementName
```

The following example finds the first h1 element in the document:

```
let firstHeading = document.querySelector('h1');
```

And the following example finds all h2 elements:

```
let heading2 = document.querySelectorAll('h2');
```

querySelector - Class Selector

To find the element with a given class attribute, you use the class selector syntax:

```
.className
```

The following example finds the first element with the menu-item class:

```
let note = document.querySelector('.menu-item');
```

And the following example finds all elements with the menu class:

```
let notes = document.querySelectorAll('.menu-item');
```

querySelector - ID Selector

To select an element based on the value of its id, you use the id selector syntax:

```
#id
```

The following example finds the first element with the id #logo:

```
let logo = document.querySelector('#logo');
```

Since the id should be unique in the document, the querySelectorAll() is not relevant.

querySelector - Attribute Selector

To select all elements that have a given attribute, you use one of the following attribute selector syntaxes

```
[attribute]
[attribute=value]
```

The following example finds the first element with the attribute [autoplay] with any value:

```
let autoplay = document.querySelector('[autoplay]');
```

And the following example finds all elements that have [autoplay] attribute with any value:

```
let autoplays = document.querySelectorAll('[autoplay]');
```

querySelector - Grouping Selectors

To group multiple selectors, you use the following syntax:

```
selector, selector, ...
```

The selector list will match any element with one of the selectors in the group.

The following example finds all and <p> elements:

```
let elements = document.querySelectorAll('<div>, <p>');
```

querySelector - Child Combinator

The > child combinator finds all elements that are direct children of the first element:

```
selector > selector
```

The following example finds all li elements that are directly inside a <ul> element:

```
let listItems = document.querySelectorAll('ul > li');
```

To select all li elements that are directly inside a <ul> element with the class nav:

```
let listItems = document.querySelectorAll('ul.nav > li');
```

querySelector - Descendant Cominator

To find descendants of a node, you use the space ( ) descendant combinator syntax:

```
selector selector
```

For example p a will match all <a> elements inside the p element:

```
let links = document.querySelector('p a');
```