WMNFE 2210

FRONT-END DEVELOPMENT

JavaScript #9 | DOM Navigation & Manipulation

CREATIVE
MEDIA
EDUCATION

ΠΕΡΙΕΧΟΜΕΝΑ

Περιεχόμενα

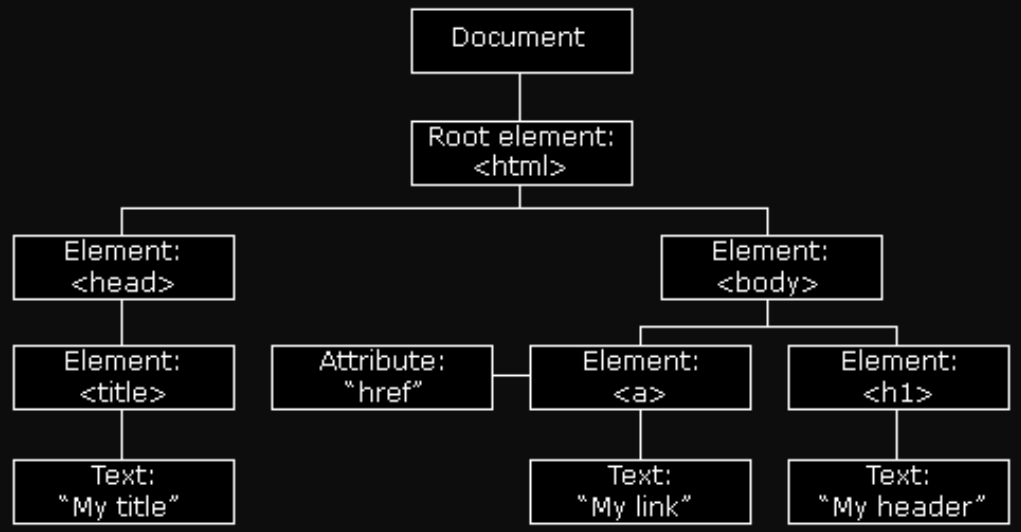
- **DOM** Navigation
  - Accessing child nodes
  - Accessing parent nodes
  - Accessing sibling nodes
- **DOM** Manipulation
  - Adding new elements
  - Accessing / setting HTML content
  - Removing existing elements
  - Replacing existing elements
  - Accessing / setting attributes

DOM

DOM

HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a **Document Object Model** of the page, as a tree of elements:



DOM

HTML DOM API

The **HTML DOM API** is made up of the interfaces that define the functionality of each of the elements in **HTML**, as well as any supporting types and interfaces they rely upon.

This **API** allows JavaScript to:

- change all the **HTML** elements in the page
- change all the **HTML** attributes in the page
- change all the **CSS** styles in the page
- remove existing **HTML** elements and attributes
- add new **HTML** elements and attributes
- react to all existing **HTML** events in the page
- create new **HTML** events in the page

DOM

Types of DOM Nodes

The **DOM** tree is consists of different types of nodes, such as elements, text, comments, etc. Every node has a nodeType property that you can use to find out what type of node you are dealing with.

Constant	Value	Description
ELEMENT_NODE	1	An element node such as **<p>** or **<img>**
TEXT_NODE	3	The actual text of element.
COMMENT_NODE	8	A comment node i.e. **<!-- some comment -->**
DOCUMENT_NODE	9	document node i.e. the parent of **<html>** element.
DOCUMENT_TYPE_NODE	10	A document type node e.g. **<!DOCTYPE html>** for HTML5 documents.

DOM NAVIGATION

DOM Navigation

Accessing child nodes

You can use the **firstChild** and **lastChild** properties of the **DOM** node to access the first and last direct child node of a node, respectively. If the node doesn't have any child element, it returns **null**.

```
<div id="main">
  <h1 id="title">Hello, World!</h1>
  <p id="hint"><span>This is a simple paragraph.</span></p>
</div>

<script>
  const main = document.getElementById('main');
  console.log(main.firstChild.nodeName); // Prints: #text

  const hint = document.getElementById('hint');
  console.log(hint.firstChild.nodeName); // Prints: SPAN
</script>
```

DOM Navigation

Text / Comment nodes

First-child node of a **DIV** element might return **#text** instead of an element.

That is because whitespace, such as spaces, tabs, newlines, etc. are valid characters and they form **#text** nodes and become a part of the **DOM** tree.

To avoid the issue with **firstChild** and **lastChild** returning **#text** or **#comment** nodes, you could alternatively use the **firstElementChild** and **lastElementChild** properties to return only the first and last element node, respectively.

DOM Navigation

Παράδειγμα

```
<div id="main">
  <h1 id="title">Hello, World!</h1>
  <p id="hint"><span>This is a simple paragraph.</span></p>
</div>

<script>
  const main = document.getElementById('main');
  console.log(main.firstElementChild.nodeName); // Outputs: H1

  const hint = document.getElementById('hint');
  console.log(hint.firstElementChild.nodeName); // Outputs: SPAN
</script>
```

DOM Navigation

All Children

Similarly, you can use the **childNodes** property to access all child nodes of a given element, where the first child node is assigned index **0**.

The **childNodes** returns all child nodes, including non-element nodes like text and comment nodes.

To get a collection of only elements, use **children** property instead.

DOM Navigation

Παράδειγμα #1

```
<div id="main">
  <h1 id="title">Hello, World!</h1>
  <p id="hint"><span>This is a simple paragraph.</span></p>
</div>

<script>
  const main = document.getElementById("main");

  // First check that the element has child nodes
  if (main.hasChildNodes()) {
    const nodes = main.childNodes;

    // Loop through node list and display node name
    for (const node in nodes) {
      console.log(node.nodeName);
    }
  }
</script>
```

DOM Navigation

Παράδειγμα #2

```
<div id="main">
  <h1 id="title">Hello, World!</h1>
  <p id="hint"><span>This is a simple paragraph.</span></p>
</div>

<script>
  const main = document.getElementById("main");

  // First check that the element has child nodes
  if (main.hasChildNodes()) {
    const nodes = main.children;

    // Loop through node list and display node name
    for (const node in nodes) {
      console.log(node.nodeName);
    }
  }
</script>
```

DOM Navigation

Accessing parent nodes

You can use the **parentNode** property to access the parent of the specified node in the **DOM** tree.

The **parentNode** will always return **null** for document node, since it doesn't have a parent.

However, if you want to get only element nodes you can use the **parentElement**.

DOM Navigation

Παράδειγμα #1

```
<div id="main">
  <h1 id="title">Hello, World!</h1>
  <p id="hint"><span>This is a simple paragraph.</span></p>
</div>

<script>
  const hint = document.getElementById("hint");
  console.log(hint.parentNode.nodeName); // Outputs: DIV
</script>
```

DOM Navigation

Παράδειγμα #2

```
<script>
  console.log(document);                                          // Outputs: #document
  console.log(document.documentElement.parentNode.nodeName);      // Outputs: #document
  console.log(document.documentElement.nodeName);                 // Outputs: HTML
  console.log(document.parentNode);                               // Outputs: null

  // This is, possibly, the only case
  // where you parentNode and parentElement are different
  console.log(document.documentElement.parentNode)
  console.log(document.documentElement.parentElement)
</script>
```

DOM Navigation

Accessing sibling nodes

You can use the **previousSibling** and **nextSibling** properties to access the previous and next node in the **DOM tree**, respectively.

Alternatively, you can use the **previousElementSibling** and **nextElementSibling** to get the previous and next sibling element skipping any whitespace text nodes.

All these properties returns **null** if there is no such sibling.

DOM Navigation

Παράδειγμα #1

```
<div id="main">
  <h1 id="title">Hello, World!</h1>
  <p id="hint"><span>This is a simple paragraph.</span></p><hr>
</div>

<script>
  const title = document.getElementById("title");
  console.log(title.previousSibling.nodeName); // Outputs: #text

  const hint = document.getElementById("hint");
  console.log(hint.nextSibling.nodeName);      // Outputs: HR
</script>
```

DOM Navigation

Παράδειγμα #2

```
<div id="main">
  <h1 id="title">Hello, World!</h1>
  <p id="hint"><span>This is a simple paragraph.</span></p>
</div>

<script>
  const hint = document.getElementById('hint');
  console.log(hint.previousElementSibling.nodeName);     // Outputs: H1
  console.log(hint.previousElementSibling.textContent); // Outputs: Hello, World!

  const title = document.getElementById('title');
  console.log(title.nextElementSibling.nodeName);        // Outputs: P
  console.log(title.nextElementSibling.textContent);     // Outputs: This is a simple paragra…
</script>
```

DOM MANIPULATION

DOM Manipulaton

Adding new elements

You can explicitly create new element in an **HTML** document, using the **document.createElement()** method. This method creates a new element, but it doesn't add it to the **DOM**; you'll have to do that in a separate step.

The **appendChild()** method adds the new element at the end of any other children of a specified parent node.

However, if you want to add the new element at the beginning of any other children you can use the **insertBefore()** method.

DOM Manipulaton

Παράδειγμα #1

```
<div id="main">
  <h1 id="title">Hello World!</h1>
  <p id="hint">This is a simple paragraph.</p>
</div>

<script>
  // Creating a new div element
  const newp = document.createElement('p');
  // Creating a text node
  const text = document.createTextNode('This is a new paragraph.');
  // Adding the text node to the newly created div
  newp.appendChild(text);

  // Adding the newly created element and its content into the DOM
  const main = document.getElementById('main');
  main.appendChild(newp);
</script>
```

DOM Manipulaton

Παράδειγμα #2

```
<div id="main">
  <h1 id="title">Hello World!</h1>
  <p id="hint">This is a simple paragraph.</p>
</div>

<script>
  // Creating a new div element
  const newp = document.createElement('p');
  // Creating a text node
  const text = document.createTextNode('This is a new paragraph.');
  // Adding the text node to the newly created div
  newDiv.appendChild(text);

  // Adding the newly created element and its content into the DOM
  const main = document.getElementById('main');
  const hint = document.getElementById('hint');
  main.insertBefore(newp, hint);
</script>
```

DOM Manipulaton

Getting / setting HTML content

You can also get or set the contents of the **HTML** elements easily with the **innerHTML** property. This property sets or gets the **HTML** markup contained within the element i.e. content between its opening and closing tags.

When inserting **HTML** into a page, be careful not to use user input that hasn't been escaped, to prevent **XSS** attacks.

Mote often than not, using **textContent** is a better choice than **innerHTML**, since works with plain text and not HTML.

DOM Manipulaton

Παράδειγμα #1

```
<div id="main">
  <h1 id="title">Hello World!</h1>
  <p id="hint">This is a simple paragraph.</p>
</div>

<script>
  // Getting inner HTML contents
  const contents = document.getElementById('main').innerHTML;
  console.log(contents); // Outputs inner html contents

  // Setting inner HTML contents
  const main = document.getElementById('main');
  main.innerHTML = '<p>This is <em>newly inserted</em> paragraph.</p>';
</script>
```

DOM Manipulaton

Removing existing elements

Similarly, you can use the **removeChild()** method to remove a child node from the **DOM**. This method also returns the removed node.

It is also possible to remove the child element without exactly knowing the parent element. Simply find the child element and use the **parentNode** property to find its parent element.

This property returns the parent of the specified node in the **DOM** tree.

DOM Manipulaton

Παράδειγμα #1

```
<div id="main">
  <h1 id="title">Hello World!</h1>
  <p id="hint">This is a simple paragraph.</p>
</div>

<script>
  const main = document.getElementById('main');
  const hint = document.getElementById('hint');
  main.removeChild(hint);
</script>
```

DOM Manipulaton

Παράδειγμα #2

```
<div id="main">
  <h1 id="title">Hello World!</h1>
  <p id="hint">This is a simple paragraph.</p>
</div>

<script>
  const hint = document.getElementById('hint');
  hint.parentNode.removeChild(hint);
</script>
```

DOM Manipulaton

Replacing existing elements

You can also replace an element in **HTML DOM** with another using the **replaceChild()** method.

This method accepts two parameters: the node to insert and the node to be replaced.

```
parentNode.replaceChild(newChild, oldChild);
```

DOM Manipulaton

Παράδειγμα

```
<div id="main">
  <h1 id="title">Hello World!</h1>
  <p id="hint">This is a simple paragraph.</p>
</div>

<script>
  const main = document.getElementById('main');
  const hint = document.getElementById('hint');

  // Creating new element
  const newp = document.createElement('p');
  const text = document.createTextNode('This is a new paragraph.');
  newPara.appendChild(text);

  // Replacing old paragraph with newly created paragraph
  main.replaceChild(newp, hint);
</script>
```

DOM Manipulaton

Accessing / setting attributes

You can access or set the attributes of an element using the **getAttribute()** and **setAttribute()** methods.

The **getAttribute()** method of the Element interface returns the value of a specified attribute on the element.

If the given attribute does not exist, the value returned will either be **null** or **""** (the empty string).

```
let attribute = element.getAttribute(attributeName);
```

Its complementary methods are: **setAttribute()**, **hasAttribute()** and **removeAttribute()**.

DOM Manipulaton

Element properties

An easier way to access the attributes of an element, is directly as properties of the element.

Most attributes have a corresponding attribute with the exact same name.

If the attribute's name includes *hyphens*, then the property name is the *camel-cased* version of that name.

The **class** attribute is a special case. Since it is a reserved word, the property name is **className**.

DOM Manipulaton

Παράδειγμα

```
<div id="main" class="main">
  <h1>Hello World!</h1>
  <p>
    <a href="http://www.example.com">
      <img src="img/affiliate.png">
    </a>
  </p>
</div>

<script>
  const div = document.getElementById('main');
  const a = div.querySelector('a');
  const img = div.querySelector('img');

  console.log(div.className);
  console.log(a.href);
  console.log(img.src);

  a.target = '_blank';
  img.alt = 'Affiliate link';
</script>
```

DOM Manipulaton

style property

All rules in the **style** attribute, like **property: value;**, become *"sub-properties"* of the **style** property.

```
el.style.color = 'red';
el.style.backgroundColor = 'blue';
```

The same rule, for names with *hyphens*, applies here too.

DOM Manipulaton

classList property

Although it is possible to alter styles through the **style** property, or access the **className** property directly, it is more convenient to use the **classList** property.

This property is a **DOMTokenList** object, which is a list of space-separated **CSS** classes.

You can add, remove or toggle classes using the **add()**, **remove()** and **toggle()** methods respectively, without having to worry about the existence / duplicity of class names, or splitting and concatenating them.

DOM Manipulaton

Παράδειγμα

```
<div id="main" class="black-bg">
  <h1>Hello World!</h1>
</div>

<script>
  const el = document.getElementById('main');
  el.classList.remove('black-bg');
  el.classList.add('main');

  // Poor man's version of dark mode.
  // The class name is added to the list,
  // when the second argument if `true` / `truthy`
  // otherwise it is removed, if present.
  el.classList.toggle('dark', (new Date()).getHours() > 18);
</script>
```

Χρήσιμα links

JavaScript HTML DOM
https://www.w3schools.com/js/js_htmldom.asp

JavaScript DOM Navigation - Tutorial Republic
https://www.tutorialrepublic.com/javascript-tutorial/jav…

Node.textContent - Web APIs | MDN
https://developer.mozilla.org/en-US/docs/Web/API/No…

What's Best: innerText vs. innerHTML vs. textContent | …
https://betterprogramming.pub/whats-best-innertext-vs…

Element.getAttribute() - Web APIs | MDN
https://developer.mozilla.org/en-US/docs/Web/API/Ele…

Element.classList - Web APIs | MDN
https://developer.mozilla.org/en-US/docs/Web/API/Ele…

Extra info

How to Manipulate the DOM - the Ultimate Beginner's...
https://www.freecodecamp.org/news/how-to-manipul...

What is the Difference Between textContents, innerTe...
https://www.microfocus.com/documentation/silk-test/2...

THANK YOU!