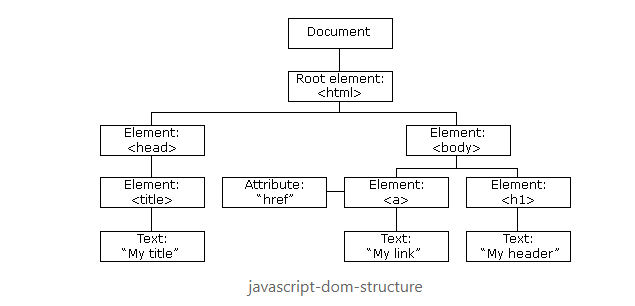
DESCRIPTION

**Document Object Model (DOM)**

The Document Object Model (DOM) is a programming API for HTML and XML documents. It enables Javascript to access and manipulate the elements and styles of a website. The browser creates a tree-like hierarchical representation of the HTML document, that tree-like structure is known as **DOM Structure** or a **DOM tree**.



Each HTML element in the DOM tree is an object. The positions of the elements in the DOM tree are nodes.The tags are element nodes. Attributes in the elements are attribute nodes. The text inside elements is a text node. It may not have children and is always a leaf of the tree. The root of the DOM tree is a <html> element, which is known as a **document object**.

**Example:**

Below, we have a simple HTML Document:

<!DOCTYPE HTML>

<html>

<head>

<title>Title goes here</title>

</head>

<body>

<p> DOM Structure </p>

</body>

</html>

The DOM structure for the above HTML document looks like:

HTML (root)

|

|---HEAD

| |

| |----TITLE

| |

| |-----#text - "Title goes here"

|

|---BODY

|

|----P

|

|----- #text - "DOM Structure"

## DESCRIPTION

# Selecting elements from the DOM

JavaScript is used to get or modify the content or value of the HTML elements on the page. To perform any action on the HTML element, we need to select the target HTML element.

The ways for selecting the elements on a page are listed below:

## Selecting Elements by ID

The getElementById() method is used to select an element based on its unique ID. The getElementById() method will return the element as an object if the matching element was found, or null if no matching element was found in the document.

**Example:** In the example below, getElementById()is a method, while innerHTML is a property. It selects the element with the id = "demo" attribute and changes the content to "Paragraph Changed".

<body>

<p id="demo">This is a paragraph.</p>

<script>

document.getElementById("demo").innerHTML = "Paragraph Changed";

</script>

</body>

## Selecting Elements by Class Name

The getElementsByClassName() method used to select all the elements having specific class names. This method returns an array-like object of all child elements which have all of the given class names.

**Example:** selects the list of elements which have class="test" attribute and changes the background color as yellow.

<body>

<div class="test"> This is a div element with class="test". </div>

<p>

<h1 class= "test"> This is a h1 element with class="test". </h1>

This is a paragraph.

</p>

<p class="test">This is a p element with class="test".</p>

<script>

var matches = document.getElementsByClassName("test");

for(var elem in matches) {

matches[elem].style.background = "yellow";

}

</script>

</body>

## Selecting Elements by Tag Name

The getElementsByTagName() method used to select HTML elements by tag name. This method also returns an array-like object of all child elements with the given tag name.

**Example:** selects all the <p> element and changes the background color as red.

<body>

<h1> Heading </h1>

<p>This is a paragraph of text.</p>

<div>This is another paragraph of text.</div>

<p>This is one more paragraph of text.</p>

<script>

var matches = document.getElementsByTagName("p");

for(var elem in matches) {

matches[elem].style.background = "red";

}

</script>

</body>

## Selecting Elements with CSS Selectors

We can use querySelector() and querySelectorAll() methods to select elements that matches the specified CSS selector. The querySelector() finds the first element that matches a CSS selector whereas the querySelectorAll() finds all elements that match a CSS selector.

**Example:**

<body>

<p id = "para">This is a paragraph</p>

<ul>

<li>Apple</li>

<li>Orange</li>

<li>Mango</li>

</ul>

<script>

var matches = document.querySelectorAll("ul li");

for(var elem of matches) {

document.write(elem.innerHTML + "<br>"); //outputs: "Apple Orange Mango"

}

document.write(document.querySelector('#para').textContent); //outputs: "This is a paragraph"

</script>

</body>

## DESCRIPTION

# Traversing the DOM

The DOM allows us to do anything with elements and their contents, but first we need to reach the corresponding DOM object. The properties used to transverse the DOM or to reach/get a particular DOM object are listed below:

## Topmost nodes

The document object is the root of every node in the DOM. The topmost tree nodes are associated with document properties:

* document.documentElement property - refers the DOM node of the <html> tag.
* document.head property - refers the DOM node of the <head> tag.
* document.body property - refers the DOM node of the <body> tag.

## Parent Nodes

The parent of any node is the node that is one level above in the DOM tree. There are two properties to get the parent — parentNode and parentElement. The parentNode is most commonly used for traversing the DOM.

**Example:**

<body>

<div id="main">

<p id="para">This is a note!</p>

</div>

<script>

let elmt = document.querySelector('#para');

document.write(elmt.parentNode+ "<br>"); // outputs: [object HTMLDivElement]

</script>

</body>

The parentElement property returns the “element” parent, whereas parentNode returns “any node” parent. With the one exception of document.documentElement:

alert( document.documentElement.parentNode ); // document

alert( document.documentElement.parentElement ); // null

## Child Nodes

The children of a node are the nodes that are one level below it. The properties are listed below:

* childNodes property - returns a list of child nodes, including text nodes.
* firstChild property - give access to the first child node.
* lastChild property - give access to the last child node.

**Example:**

<body>

<div id="myDiv">

<p>This is a paragraph - first child</p>

<div> this is a div elemt - last child</div>

</div>

<script>

let elmt = document.querySelector('#myDiv');

document.write("<br> Child nodes of div element: <br>");

for (let i = 0; i < elmt.childNodes.length; i++) {

document.write(elmt.childNodes[i] + "<br>");

}

document.write("<br> First child of div element: <br>" +elmt.firstChild) ;

document.write("<br> Last child of div element: <br>" +elmt.lastChild) ;

</script>

</body>

The output will be:

This is a paragraph - first child

this is a div elemt - last child

Child nodes of div element:

[object Text]

[object HTMLParagraphElement]

[object Text]

[object HTMLDivElement]

[object Text]

First child of div element:

[object Text]

Last child of div element:

[object Text]

## Sibling Nodes

Siblings are nodes that are children of the same parent. The siblings of a node are any node on the same tree level in the DOM.

* previousElementSibling property - give access to the previous sibling Node, i.e. the node that immediately precedes the specified node.
* nextElementSibling property - give access to the next sibling vode, i.e. the node that immediately precedes or follows the specified node.

**Example:**

<body>

<ul >

<li>list item 1</li>

<li class="list">list item 2</li>

<li>list item 3</li>

</ul>

<script>

const secondListItem = document.querySelector('.list');

document.write(secondListItem.previousElementSibling.textContent) ; // outputs: "list item 1"

document.write(secondListItem.nextElementSibling.textContent); //outputs: "list item 3"

</script>

</body>

## DESCRIPTION

# DOM Manipulation

We can add, remove, replace, and copy any element into a DOM tree. DOM Manipulation methods are listed below:

## Create Elements

The createElement() method is used to create a new element and attach it to the DOM tree.

**Example:**

var elmt = document.createElement('div');

elmt.innerHTML = '<p>Hello World!</p>';

## Replace Child Elements

The replaceChild() method is used to remove an element from the DOM tree and insert a new one in its place.

**Example:**

<body>

<div>

This is a div element.

<div>

<script>

// selecting the <div> element

var elmt = document.querySelector('div');

//creating new <p> element and adding content inside it.

var newElmt = document.createElement('p');

newElmt.innerHTML = '<b>The div element is replaced with p element</b>';

//replacing the <div> element with the <p> element

elmt.parentNode.replaceChild(newElmt, elmt);

</script>

</body>

## Remove Child Elements

The removeChild() method is used to remove an element from the DOM tree.

**Example:** Here, first we select the element to remove, then walk up the tree to its parent and remove the child element from there.

var elmt = document.querySelector('div'); // select the first returned <div> element

elmt.parentNode.removeChild(elmt);

## Append a Node

The appendChild() method is used to add a node to the end of the list of child nodes of a specified parent node.

**Example:** Here, we add three list items to the <ul> element

<body>

<ul id="animals">

</ul>

<script>

function createAnimalList(name) {

let li = document.createElement('li');

li.textContent = name;

return li;

}

// get the ul #animals

const list = document.querySelector('#animals');

// add animals to the list

list.appendChild(createAnimalList('Lion'));

list.appendChild(createAnimalList('Tiger'));

list.appendChild(createAnimalList('Wolf'));

</script>

</body>

## Insert a Node before another

The insertBefore() method is used to insert a node before another node as a child node of a parent node.

**Example:** Here, we insert the new <li> element before the first child of <ul> element.

<body>

<ul id="animal">

<li>Lion</li>

<li>Tigerr</li>

</ul>

<script>

let animal = document.getElementById('animal');

// create a new li node

let li = document.createElement('li');

li.textContent = 'Wolf';

// insert a new node before the first list item

animal.insertBefore(li, animal.firstElementChild);

</script>

</body>

## Insert a Node After another

JavaScript DOM provides the insertBefore() method that allows you to insert a new after an existing node as a child node. However, it hasn’t supported the insertAfter() method yet.

So, we can insert a new node after an existing node as a child node, by selecting a parent node of the existing node and call the insertBefore() method on the parent node to insert a new node before that immediate sibling node.

**Example:** Here, we inserts the new <li> element after the first child of <ul> element.

<body>

<ul id="animal">

<li>Lion</li>

<li>Tiger</li>

</ul>

<script>

let animal = document.getElementById('animal');

// create a new li node

let li = document.createElement('li');

li.textContent = 'Wolf';

// insert a new node before the first list item

animal.insertBefore(li, animal.firstElementChild.nextSibling);

</script>

</body>

## Get or Set text of a Node

The textContent property is used to get and set a text content inside a particular node.

**Example:**

<body>

<div id = "content">

This is div element.

</div>

<script>

// Getting a text content

let content = document.getElementById('content');

alert("Getting a text content inside div element: " + content.textContent);

//setting a text content

content.textContent = 'New text content in the div element';

</script>

</body>

## Get or Set HTML of Element

The innerHTML property to get the text and inner HTML elements inside a the particular element and the setting will replace the existing text and inner tags inside an element with the new content.

**innerHTML vs textContent** - The innerHTML property returns the text and inner HTML elements. The textContent property returns only the text Content.

**Example:**

<body>

<div id="myBdy">

<p id = "para">This is Paragraph.</p>

<button onclick="myFunction()" >Try it</button>

<p id="demo"></p>

</div>

<script>

function myFunction() {

// get HTML of Element

var x = document.getElementById("para").innerHTML;

document.getElementById("demo").innerHTML = x;

}

// You can understand the difference between innerHTML and textContent property clearly from the output of the

// below two alert boxes

alert ("textcontent property:" + document.getElementById("myBdy").textContent);

alert ("innerHTML property:" + document.getElementById("myBdy").innerHTML);

</script>

</body>

## Clone a Node

The cloneNode() method is used to clone an element. The cloneNode() method accepts an optional parameter. If the parameter value is true, then the original node and all of its descendants are cloned. If the parameter value is false, only the original node will be cloned.

**Example:** Here, the parameter value for cloneNode() method is true. So it clones the target node and all of its descendants.

<body>

<ul id="animal">

<li>Lion</li>

<li>Tiger</li>

<li>Wolf</li>

</ul>

<script>

let list = document.querySelector('#animal');

let clonedList = list.cloneNode(true);

clonedList.id = 'cloned animal';

document.body.appendChild(clonedList);

</script>

</body>

## Managing Attributes

* getAttribute(attribute\_name) method - Used to get the value of an attribute on a specified element
* setAttribute(attribute\_name, attribute\_value) method - Used to set a value of an attribute on a specified element,
* removeAttribute(attribute\_name) method - Used to remove an attribute with a specified name from an element
* hasAttribute(attribute\_name) method - Used to check an element has a specified attribute or not.

DESCRIPTION

**JavaScript Events**

Events occur when user interaction takes place on a web page, such as when the end-user clicked a link or button, pressed a key on the keyboard, moved the mouse pointer, submits a form, etc. The browser also triggers the events, such as the page load and unload events.

When an event occurs, we use a JavaScript event handler (or an event listener) to detect them and perform a specific task.

Some of the commonly used JavaScript Events are listed below:

* **onclick** - This is a click event occurs when a user clicks the element on a web page.
* **ondblclick** - This is a click event occurs when a user double clicks the element on a web page.
* **onload** - This is a load event occurs when the browser has finished loading the page.
* **onunload** - This is a load event occurs when a user closes the document.
* **onresize** - This is a load event occurs when the browser window is minimized or maximized.
* **onmouseover** - This is a mouse event occurs when the user moves the mouse over an HTML element.
* **onmouseout** - This is a mouse event occurs when the user moves the mouse away from an HTML element.
* **onkeydown** - This is a keyboard event occurs when the user presses down a key on the keyboard.
* **onkeyup** - This is a keyboard event occurs when the user releases a key on the keyboard.
* **onfocus** - This is a form-input event occurs when the user gives focus to an element on a web page.
* **onblur** - This is a form-input event occurs when the user takes the focus away from a form element or a window.
* **onchange** - This is a form-input event that occurs when a user changes the value of a form element.
* **onsubmit** - This is a form-input event that occurs when the user submits a form on a web page.

**Example:**

<body>

<button onclick= onclickEvent()>Click me!!</button>

<p id= "para" onmouseover = onmouseoverEvent() onmouseout = onmouseoutEvent() >This is a Paragraph....</p>

<script>

function onclickEvent(){

alert('Hello, You clicked the button');

}

function onmouseoverEvent(){

document.getElementById("para").style.backgroundColor = "green";

}

function onmouseoutEvent(){

document.getElementById("para").style.backgroundColor = "yellow";

}

</script>

</body>

**JavaScript EventListener**

An event listener is a function in JavaScript that waits for an event to occur. The addEventListener() function is an inbuilt function in JavaScript used to handle the event.

The Syntax of addEventListener() function: element.addEventListener(event, function, useCapture)

Where,

* event - Specifies the name of the event.
* function - Specifies the function to run when the event occurs
* useCapture - It is an optional parameter takes a boolean value. If the parameter value is true then the event handler is executed in the capturing phase. If the parameter value is false then the event handler is executed in the bubbling phase.

The removeEventListener() method used to remove an event handler that has been attached with the addEventListener() method.

**Example:**

<body>

<h2>JavaScript addEventListener()</h2>

<p id="myBtn">This is a paragraph.</p>

<p id="demo"></p>

<script>

var x = document.getElementById("myBtn");

x.addEventListener("mouseover", myFunction);

x.addEventListener("mouseout", mySecondFunction);

function myFunction() {

document.getElementById("demo").innerHTML += "Moused over!<br>";

x.style.backgroundColor = "green";

}

function mySecondFunction() {

document.getElementById("demo").innerHTML += "Moused out!<br>";

x.style.backgroundColor = "white";

}

</script>

</body>