Contents

[Window object 2](#_Toc110711301)

[DOM (Document Object Model) 2](#_Toc110711302)

[Document object basics 3](#_Toc110711303)

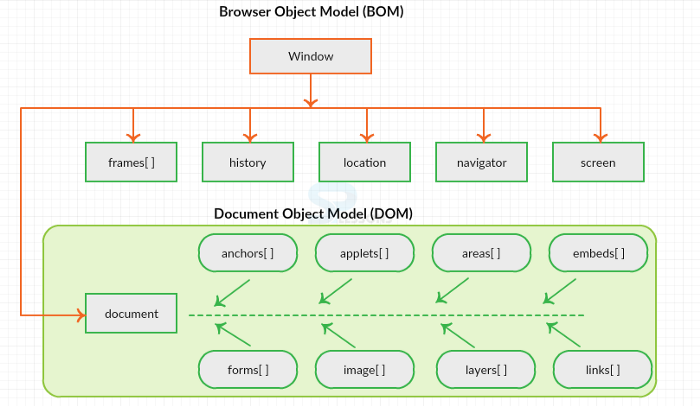
[Document query selector 4](#_Toc110711304)

[DOM Traversing 6](#_Toc110711305)

[Create element runtime and add to existing element in DOM 7](#_Toc110711306)

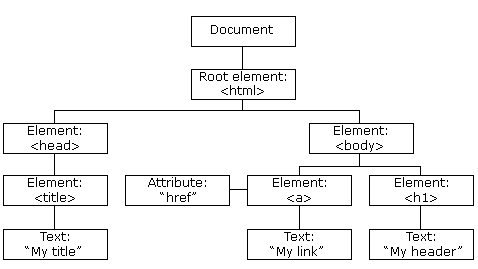
# Window object

This is the root most object in browser tab. iFrame inside windows also has its own windows tag, but window object signifies the browser scree/tab



# DOM (Document Object Model)

When page loads in Window DOM is created. DOM represents objects in webpage as tree and has object oriented representation



## Document object basics

Document object can be used in JavaScript to access basic info around DOM. Following list is just few examples and document object can be used to access ay more properties of DOM.

let val ;

// selects whole document object

val= document;

//selects individual head tag in DOM

val = document.head;

// selects body tag in DOM

val = document.body;

// returns URL from browser

val= document.URL;

// returns domain or website

val = document.domain;

console.log(val);

We can access individual items in DOM like images, scripts, links etc.. returned list has individual objects having crazy amount of data in them.

// access list of all images from DOM

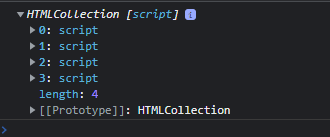
val=document.images; // returns all image objects in a list

// get list of all the scripts in <script> tag

val = document.scripts;

console.log(val);

returned list in above example is HTMLCollection and we can’t directly loop in it. Below is screenshot from console.



// we can access individual item and its property from returned object

console.log(val[0].src)

For looping through HTMLCollection we need to convert it to array first and then we can get attributes of individual element.

scripts = document.scripts;

// document.scripts returns HTMLCollection which can’t be looped using foreach

// for looping converting collection to array

let scriptArr= Array.from(scripts);

// looping through individual items and accessing its attributes

scriptArr.forEach(function(script) {

  console.log(script.getAttribute('src'));

});

## Document query selector

Document.querySelector(“<css selector>”) : Returns the first element in DOM that matched provided CSS selector.

Document.querySelectorAll(“<css selector>”) : Returns all element in DOM that matched provided CSS selector.

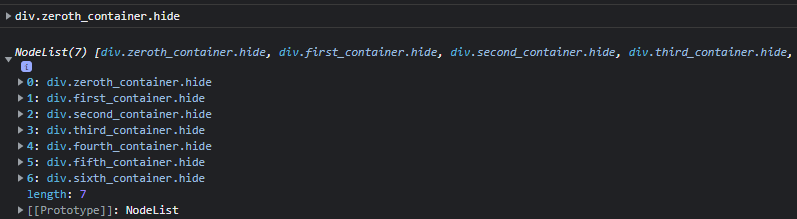
let test = document.querySelector(".hide")

console.log(test);

let testAll = document.querySelectorAll(".hide")

console.log(testAll);

Below results shows first single return and second NodeList of all matching elements



Returned nodelist can be directly looped and we can get or set any details of child elements.

Accessing child element directly using index [1],[3] is not recommendable since if HTML code changes then Javascript will start failing as elements position in nodelist might change. Belo code is just for example.

let test = document.querySelector(".hide")

console.log(test);

let testAll = document.querySelectorAll(".hide")

console.log(testAll);

// return is nodelist from queryselectorAll so we can loop directly

// without converting to array and add attribute selection to find what we need.

// Ex. in below code we are finding image src URL's for all returned items

testAll.forEach(function(item){

    // we can now process individual item having class as .hide.

    // get list of child elements of item having class as .hide

    console.log(item.childNodes);

    // in child elements item at position 3 is <p> so we can access it

    // and change its attributes of colour to anything

    console.log(item.childNodes[3].style.color= 'red');

    // we can also read value of any attribute from items

    console.log(item.childNodes[1].getAttribute("src"));

})

// in above example we can also use item.children. The main difference between

// children and childNodes property is that children work upon elements and childNodes

// on nodes including non-element nodes like text and comment nodes.

Better way of updating colour for elements by hardcoding their position we can use css selector like

//instead of updating item style as mentioned in above code by hardcoding item position

//we can directly get all the required <p> tags and update the style without hardcoding it

// below selector is getting <p> tags from parents with class name \*\_container

let elements = document.querySelectorAll('div[class\*="\_container"] p');

elements.forEach(function(item){

    item.style.color= 'blue';

})

Selecting elements with different methods than QuerySelector

// There are different ways as well we can select DOM using name, Id etc.

// and perform different operations

let newval = document.getElementsByName("second\_container\_name")

newval[0].innerHTML = "<h1> Changed Text </h1>";

## DOM Traversing

We can move inside DOM up (parent), down(children), side(sibling) using different methods.

if we use ‘element’ named method then it usually gives real element rather than fake ‘text’, ‘comment’ elements what we get. Ex. FirstChild vs FirstElementChild, nextSibling vs nextElementSibling etc…

Alo below example shows use of element.children method rather than element.childNodes.

Element.children is more preferred method

// instead of childNodes we can use children method as well to get

// child elements

let elementAll = document.querySelectorAll(".hide")

elementAll.forEach(function(element){

    let val = element.children;

    // this returns HTMLCollection and notNodeList like childNodes function

    //console.log(val[1].innerHTML);

    // item at position 1 is <p> so we have changed its internal text to 'updated text'

    val[1].innerHTML = "<h2> updated text </h2>";

    // get next element in DOM.

    console.log(`Next Sibling: ${element.nextSibling.innerHTML}`); // returns text

    console.log(`Next Element Sibling: ${element.nextElementSibling.innerHTML}`); // return actual element

    //  get previous element in DOM.

    console.log(`Previous Sibling: ${element.previousSibling.innerHTML}`); // returns text

    console.log(`Previous Element Element: ${element.previousElementSibling.innerHTML}`); // return actual element

    // like child we can also get parent element details using ParentNode or ParentElement

    console.log(`Parent Element: ${element.parentNode.firstElementChild.innerHTML}`);

    console.log(`Parent Element: ${element.parentElement.innerHTML}`);

})

## Create element runtime and add to existing element in DOM

We need to keep creating new child elements and keep adding properties/text to mimic how we create elements in html

/\*

We need to create following item in DOM runtime and add it under <div class="getBoundingClientRect">

<div class="tength\_container hide">

<img src="https://source.unsplash.com/random/?dark-technology" alt="technology">

<p>

    tength : Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse ultrices viverra tortor, eu consequat urna porta nec. Duis risus lorem, blandit sed nulla non, consequat porta justo. Nulla feugiat vulputate sagittis. Aliquam vel odio enim. Donec id maximus ante, nec vestibulum urna. Nulla pulvinar iaculis nibh, in vestibulum neque commodo ac. Fusce ac arcu vel felis lobortis hendrerit et a libero. Duis non arcu ex. Suspendisse tempor lectus ac arcu ultrices, vel semper lorem commodo. Donec faucibus vel turpis in ultrices.

</p>

</div>

\*/

//new div tag

const mainElement = document.createElement('div');

// add class name to div tag

mainElement.className = 'tength\_container hide';

// create image tag with src and alt attributes

const image = document.createElement('img');

image.setAttribute('src','https://source.unsplash.com/random/?dark-technology');

image.setAttribute('alt','technology');

//create p tag with text contents

const para = document.createElement('p');

para.textContent = "tength : Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse ultrices viverra tortor, eu consequat urna porta nec. Duis risus lorem, blandit sed nulla non, consequat porta justo. Nulla feugiat vulputate sagittis. Aliquam vel odio enim. Donec id maximus ante, nec vestibulum urna. Nulla pulvinar iaculis nibh, in vestibulum neque commodo ac. Fusce ac arcu vel felis lobortis hendrerit et a libero. Duis non arcu ex. Suspendisse tempor lectus ac arcu ultrices, vel semper lorem commodo. Donec faucibus vel turpis in ultrices."

//add image and p tag to parent div

mainElement.appendChild(image);

mainElement.appendChild(para);

console.log(`create item main\_element : ${mainElement.innerHTML}`);

const parentElement = document.querySelector(".getBoundingClientRect");

// add runtime created div element to existing element with class name .getBoundingClientRect"

parentElement.appendChild(mainElement);