What is the DOM?

The DOM is a W3C (World Wide Web Consortium) standard. The DOM defines a standard for accessing documents. The W3C DOM standard is separated into 3 different parts:

- Core DOM - standard model for all document types
- XML DOM - standard model for XML documents
- HTML DOM - standard model for HTML documents

The HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

With the HTML DOM, JavaScript can access and change all the elements of an HTML document.

The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:

- The HTML elements as **objects**
- The **properties** of all HTML elements
- The **methods** to access all HTML elements
- The **events** for all HTML elements

In other words: **The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**

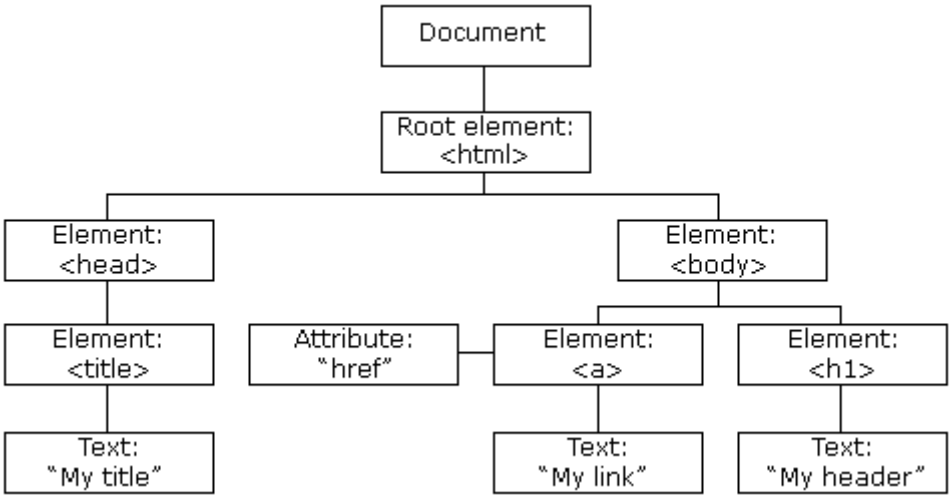
With the object model, JavaScript gets all the power it needs to create dynamic HTML:

1. JavaScript can change
   a. all the HTML elements in the page
   b. all the HTML attributes in the page
   c. all the CSS styles in the page

2. Further, JavaScript can
   a. remove existing HTML elements and attributes
   b. add new HTML elements and attributes
   c. react to all existing HTML events in the page
   d. can create new HTML events in the page

The **HTML DOM** model is constructed as a tree of **Objects**:

The HTML DOM Tree of Objects



HTML DOM Methods

HTML DOM methods are **actions** you can perform (on HTML Elements).

HTML DOM properties are **values** (of HTML Elements) that you can set or change.

The DOM Programming Interface

The HTML DOM can be accessed with JavaScript (and also with other programming languages).

In the DOM, all HTML elements are defined as **objects**. The programming interface is the properties and methods of each object.

A **property** is a value that you can get or set (like changing the content of an HTML element).

A **method** is an action you can do (like add or deleting an HTML element).

Example

The following example changes the content (the `innerHTML`) of the `<p>` element with `id="demo"`:

```
<html>
     <body>
          <p id="demo"></p>
          <script>
            document.getElementById("demo").innerHTML = "Hello World!";
          </script>
     </body>
</html>
```

In the example above, `getElementById` is a **method**, while `innerHTML` is a **property**.

The getElementById Method

The most common way to access an HTML element is to use the `id` of the element.

In the example above the `getElementById` method used `id="demo"` to find the element.

The innerHTML Property

The easiest way to get the content of an element is by using the innerHTML property. The `innerHTML` property is useful for getting or replacing the content of HTML elements.

This property can be used to get or change any HTML element, including `<html>` and `<body>`.

HTML DOM Document Object

The HTML DOM document object is the owner of all other objects in your web page. The document object represents your web page.

If you want to access any element in an HTML page, you always start with accessing the document object.

Below are some examples of how you can use the document object to access and manipulate HTML.

Finding HTML Elements

Method	Description
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

Changing HTML Elements

Property	Description
element.innerHTML = new html content	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.style.property = new style	Change the style of an HTML element
Method	Description
element.setAttribute(attribute, value)	Change the attribute value of an HTML element

Adding and Deleting Elements

Method	Description
document.createElement(element)	Create an HTML element
document.removeChild(element)	Remove an HTML element
document.appendChild(element)	Add an HTML element
document.replaceChild(new, old)	Replace an HTML element
document.write(text)	Write into the HTML output stream

Adding Events Handlers

Method	Description
document.getElementById(*id*).onclick = function(){*code*}	Adding event handler code to an onclick event

Finding HTML Objects

Property	Description
document.baseURI	Returns the absolute base URI of the document
document.body	Returns the <body> element
document.cookie	Returns the document's cookie
document.doctype	Returns the document's doctype
document.documentElement	Returns the <html> element
document.documentMode	Returns the mode used by the browser
document.documentURI	Returns the URI of the document
document.embeds	Returns all <embed> elements
document.forms	Returns all <form> elements
document.head	Returns the <head> element
document.images	Returns all <img> elements
document.implementation	Returns the DOM implementation
document.inputEncoding	Returns the document's encoding (character set)
document.lastModified	Returns the date and time the document was updated
document.links	Returns all <area> and <a> elements that have a href attribute

document.readyState	Returns the (loading) status of the document
document.referrer	Returns the URI of the referrer (the linking document)
document.scripts	Returns all <script> elements
document.strictErrorChecking	Returns if error checking is enforced
document.title	Returns the <title> element
document.URL	Returns the complete URL of the document

Finding HTML Element by Id

The easiest way to find an HTML element in the DOM, is by using the element id.

This example finds the element with `id="intro"`:

Example

```
const element = document.getElementById("intro");
```

If the element is found, the method will return the element as an object (in element).

If the element is not found, element will contain `null`.

Finding HTML Elements by Tag Name

This example finds all `<p>` elements:

Example

```
const element = document.getElementsByTagName("p");
```

```
const x = document.getElementById("main");
const y = x.getElementsByTagName("p");
```

Above example finds the element with `id="main"`, and then finds all `<p>` elements inside `"main"`

Finding HTML Elements by Class Name

If you want to find all HTML elements with the same class name, use `getElementsByClassName()`.

This example returns a list of all elements with `class="intro"`.

Example

```
const x = document.getElementsByClassName("intro");
```

Finding HTML Elements by CSS Selectors

If you want to find all HTML elements that match a specified CSS selector (id, class names, types, attributes, values of attributes, etc), use the `querySelectorAll()` method.

This example returns a list of all `<p>` elements with `class="intro"`.

Example

```
const x = document.querySelectorAll("p.intro");
```

Finding HTML Elements by HTML Object Collections

This example finds the form element with `id="frm1"`, in the forms collection, and displays all element values:

Example

```
const x = document.forms["frm1"];
let text = "";
for (let i = 0; i < x.length; i++) {
  text += x.elements[i].value + "<br>";
}
document.getElementById("demo").innerHTML = text;
```

JavaScript HTML DOM - Changing HTML

The HTML DOM allows JavaScript to change the content of HTML elements.

The easiest way to modify the content of an HTML element is by using the `innerHTML` property.

```
document.getElementById(id).innerHTML = new HTML
```

This example changes the content of a `<p>` element:

Example

```
<html>
<body>

<p id="p1">Hello World!</p>

<script>
document.getElementById("p1").innerHTML = "New text!";
</script>

</body>
</html>
```

Explaination:

- The HTML document above contains a `<p>` element with `id="p1"`
- We use the HTML DOM to get the element with `id="p1"`
- A JavaScript changes the content (`innerHTML`) of that element to "New text!"

This example changes the content of an `<h1>` element:

Example

```
<!DOCTYPE html>
<html>
<body>
<h1 id="id01">Old Heading</h1>
<script>
const element = document.getElementById("id01");
element.innerHTML = "New Heading";
</script>
```

```
</body>
</html>
```

Explanation:

- The HTML document above contains an `<h1>` element with `id="id01"`
- We use the HTML DOM to get the element with `id="id01"`
- A JavaScript changes the content (`innerHTML`) of that element to "New Heading"

Changing the Value of an Attribute

To change the value of an HTML attribute, use this syntax:

```
document.getElementById(id).attribute = new value
```

This example changes the value of the src attribute of an `<img>` element:

Example

```
<!DOCTYPE html>
<html>
<body>
<img id="myImage" src="smiley.gif">
<script>
document.getElementById("myImage").src = "landscape.jpg";
</script>
</body>
</html>
```

Example explained:

- The HTML document above contains an `<img>` element with `id="myImage"`
- We use the HTML DOM to get the element with `id="myImage"`
- A JavaScript changes the `src` attribute of that element from "smiley.gif" to "landscape.jpg"

Dynamic HTML content

JavaScript can create dynamic HTML content:

Example

```
<!DOCTYPE html>
<html>
<body>
<script>
document.getElementById("demo").innerHTML = "Date : " +
```

```
Date(); </script>
</body>
</html>
```

document.write()

In JavaScript, `document.write()` can be used to write directly to the HTML output stream:

Example

```
<!DOCTYPE html>
<html>
<body>

<p>Bla bla bla</p>

<script>
document.write(Date());
</script>

<p>Bla bla bla</p>

</body>
</html>
```

Never use `document.write()` after the document is loaded. It will overwrite the document.