JavaScript DOM (Document Object Model)

The **DOM (Document Object Model)** is a programming interface for web documents. It represents the structure of a document as a tree of nodes, where each node corresponds to a part of the document (such as an element, attribute, or text). The DOM provides a way for JavaScript to interact with HTML and XML documents, allowing developers to dynamically manipulate the content, structure, and style of a webpage.

Key Concepts of the DOM

- **Document** - Represents the entire web page as an object. It provides methods to access elements, modify content, and interact with the page.
- **Nodes** - Everything in the DOM (elements, attributes, text, etc.) is a node. The main node types are elements, text, and attributes.
- **Elements** - HTML tags, such as `<div>`, `<p>`, `<a>`, etc., are represented as element nodes in the DOM tree.
- **Attributes** - HTML attributes, such as `class`, `id`, and `href`, are represented as attribute nodes.
- **Text** - The text content inside elements is represented as text nodes in the DOM.

Manipulating the DOM

JavaScript provides a variety of methods to interact with and modify the DOM:

- **Selecting elements:** You can select HTML elements using methods such as `document.getElementById()`, `document.querySelector()`, and `document.querySelectorAll()`.
- **Modifying content:** You can change the content of an element by manipulating properties like `innerHTML`, `textContent`, or `value`.
- **Modifying attributes:** You can change an element's attributes using `setAttribute()` or directly accessing properties like `className` or `src`.
- **Adding/removing elements:** You can create new elements with `document.createElement()` and insert them into the DOM using methods like `appendChild()`, `insertBefore()`, or `removeChild()`.
- **Event handling:** The DOM allows you to register event listeners to elements to respond to user interactions like clicks, key presses, and mouse movements using `addEventListener()`.

Common DOM Methods

- **document.getElementById()** - Selects an element by its unique `id` attribute.
- **document.querySelector()** - Selects the first matching element based on a CSS selector.
- **document.querySelectorAll()** - Selects all matching elements based on a CSS selector.
- **element.addEventListener()** - Registers an event listener on an element.
- **element.innerHTML** - Gets or sets the HTML content inside an element.
- **element.textContent** - Gets or sets the text content inside an element.
- **element.setAttribute()** - Modifies an element's attribute value.
- **document.createElement()** - Creates a new HTML element.
- **element.appendChild()** - Adds a new child element to an existing element.

The **Document Object Model (DOM)** allows JavaScript to dynamically access and manipulate the structure, content, and style of a webpage.

```
    <div id='header'>The header</div>
    <div>
        <ul id="myList">
            <li id="first">item1</li>
            <li id="second">item2</li>
            <li id="third">item3</li>
        </ul>
    </div>
    <div id='footer'>The footer</div>
</div>
```

1. Accessing DOM Elements

Methods to Access Elements

Using `getElementById`

```
<script>
    let header = document.getElementById('header');
    let footer = document.getElementById('footer');
    console.log(header.innerText); // Output: The header
    console.log(footer.innerText); // Output: The footer
</script>
```

Using `getElementsByTagName`

```
<script>
    let divs = document.getElementsByTagName('div');
    console.log(divs[0].innerText); // Output: The header
    console.log(divs[2].innerText); // Output: The footer
</script>
```

Using `querySelector` (selects first match)

```
<script>
    let li1 = document.querySelector('li');
    let second = document.querySelector('li:last-of-type');
    console.log(li1.innerText); // Output: item1
    console.log(second.innerText); // Output: item3
</script>
```

Using `querySelectorAll` (selects all matches)

```
<script>
    let items = document.querySelectorAll('li');
    console.log(items[0].innerText); // Output: item1
    console.log(items[1].innerText); // Output: item2
    console.log(items[2].innerText); // Output: item3
</script>
```

Child Nodes

```
<script>
    let parentDiv = document.getElementById('container');
    let children = parentDiv.children;
    console.log(children);
</script>
```

First and Last Child

```
<script>
    let firstChild = parentDiv.firstChild;
    let lastChild = parentDiv.lastChild;
    console.log(firstChild);
    console.log(lastChild);
</script>
```

Navigating Between Siblings

```
<script>
    let secondChild = firstChild.nextSibling;
    console.log(secondChild.parentElement);
    console.log(lastChild.previousElementSibling);
</script>
```

3. Manipulating the DOM

Adding Elements Dynamically

```
<script>
    var dynamicDiv = document.createElement('div');
    var span = document.createElement('span');
    span.innerHTML = 'A new span! Dynamically added';
    dynamicDiv.appendChild(span);
    document.body.appendChild(dynamicDiv);
</script>
```

Inserting a New Text Node Before Footer

```
<script>
    let footer = document.getElementById('footer');
    let newTextNode = document.createTextNode("This is a new text node before the footer.");
    document.getElementById('container').insertBefore(newTextNode, footer);
</script>
```

Removing Elements

```
    dynamicDiv.remove();
</script>
```

4. Understanding Events and Event Handling in JavaScript

In JavaScript, events are actions or occurrences that happen in the system you are working with. These actions can be triggered by the user (like clicking a button or typing in an input field), the browser (like page load), or other parts of the web application (like a timer or network request). Some of the most common events include:

- **click** - Triggered when an element is clicked.
- **mouseover** - Triggered when the mouse pointer moves over an element.
- **keydown** - Triggered when a key is pressed down.
- **submit** - Triggered when a form is submitted.
- **load** - Triggered when a page or image is fully loaded.
- **focus** - Triggered when an element gains focus (e.g., a form field).
- **resize** - Triggered when the window is resized.

What is Event Handling?

Event handling refers to the process of responding to events in your application. In JavaScript, you can define functions that run when a particular event occurs, known as event listeners. These event listeners can be attached to DOM elements to perform certain actions in response to user interaction or other events.

When an event occurs, the browser creates an **event object** that contains all the information about the event, such as the type of event (e.g., "click"), the target element (the element that triggered the event), and other details specific to that event. The event object is passed as a parameter to the event handler function.

Attaching Event Handlers

There are multiple ways to attach event handlers in JavaScript:

- **Inline Event Handlers:** Adding the event handler directly in the HTML element's attribute (e.g., `<button onclick="alert('Hello!')">Click Me</button>`), though this approach is not recommended due to separation of concerns.
- **DOM Level 0 Event Handlers:** Assigning a function to an element's event property (e.g., `element.onclick = myFunction;`).
- **DOM Level 2 Event Handlers:** Using `addEventListener()` (modern browsers) or `attachEvent()` (for older IE versions) to add multiple event listeners to an element.

1. Inline Event Handler

```
<button onclick="alert('Hello!')">Click Me</button>
```

Click Me

This example uses an inline event handler. When you click the button, an alert box pops up saying "Hello!". This is a direct implementation of JavaScript within the HTML.

Note: This approach is generally not recommended due to the violation of the separation of concerns.

```
<button id="clickme">Click Me</button>
<script>
    // DOM Level 0 event handler
    var button = document.getElementById('clickme');
    button.onclick = function() {
        alert('Button was clicked!');
    };
</script>
```

Click Me

This example demonstrates the DOM Level 0 event handler. The `onclick` property of the button element is assigned a function that triggers an alert when clicked. This approach allows for separation of concerns but can only attach one event handler per element for each event type.

---

3. DOM Level 2 Event Handler

```
<button id="clickme2">Click Me</button>
<script>
    // DOM Level 2 event handler (addEventListener for modern browsers)
    var button = document.getElementById('clickme2');
    button.addEventListener('click', function() {
        alert('Button clicked using addEventListener!');
    });
</script>
```

Click Me

This example uses the DOM Level 2 event handler method with `addEventListener` for modern browsers and `attachEvent` for older versions of Internet Explorer. This approach allows you to add multiple event listeners to the same element and provides better flexibility.

Advantages: Multiple listeners can be attached for the same event. It also supports modern and older browsers.

Event Bubbling and Capturing

JavaScript events follow a concept called "event propagation," which consists of two phases:

- **Bubbling:** The event starts from the target element and "bubbles up" through the ancestors (parent elements) until it reaches the root of the document.
- **Capturing:** The event starts from the root of the document and travels down to the target element.

By default, events propagate in the bubbling phase, but you can control this behavior by calling `stopPropagation()` on the event object to prevent the event from propagating further.

Preventing Default Behavior

Many events have a default behavior. For example, when a form is submitted, the browser tries to send the form data to the server. If you want to prevent this default action (like preventing a link from navigating to another page), you can use `preventDefault()` on the event object.

```
<script>
    document.querySelector("form").addEventListener("submit", function(e) {
    e.preventDefault();  // Prevent form submission
    alert("Form submission is prevented!");
    });
</script>
```

In this case, the form won't be submitted to the server when the submit button is clicked, and instead, an alert will appear.

More Examples DOM Event Handling

```
<p id="dt">Click below button to display the current date:</p>
<button id="showDate">Click me</button>
<script>
    // DOM Level 2 event handler
    var button = document.getElementById('showDate');
    button.addEventListener('click', function() {
        document.getElementById('dt').innerHTML = new Date();
    });
</script>
```

To display the current date, writing above code in html file and try it yourself

Write below code in html file and try it yourself

```
<h2>Interactive List</h2>
<ul id="list">
    <li id="li1">item1</li>
    <li id="li2">item2</li>
    <li id="li3">item3</li>
</ul>
<script>
    // Adding Event Listeners for the List Items
    document.getElementById("li1").addEventListener("click", changeColor);

    // Function to change the color of item1 when clicked
    function changeColor() {
        document.getElementById("li1").style.backgroundColor = "red";
    }
    // Adding Event Listeners for the List Items
    document.getElementById("li2").addEventListener("mouseover", changeTextColor);
    // Function to change the text color of item2 when mouse is over
    function changeTextColor() {
        document.getElementById("li2").style.color = "blue";
    }
    //Removing last list item dynamically
    const li3 = document.getElementById("li3");
    li3.addEventListener("click",()=>{
        li3.remove();
```

.

Login Form

Write below code in html file and try it yourself

```
<h2>Login Form</h2>
<form id="formContainer" action="#"> </form>
<button onclick="addFormFields()">Add Username and Password</button>
<script>
    // Function to add Username and Password fields after button click
    function addFormFields() {
        const formContainer = document.getElementById("formContainer");

        // Create the Username input field
        const userNameLabel = document.createElement("label");
        userNameLabel.textContent = "User Name:";
        const userNameInput = document.createElement("input");
        userNameInput.setAttribute("type", "text");
        userNameInput.setAttribute("id", "uname");
        userNameInput.setAttribute("placeholder", "Enter user Name");

        // Create the Password input field
        const passwordLabel = document.createElement("label");
        passwordLabel.textContent = "Password:";
        const passwordInput = document.createElement("input");
        passwordInput.setAttribute("type", "password");
        passwordInput.setAttribute("id", "pwd");
        passwordInput.setAttribute("placeholder", "Enter password");

        const submitBtn = document.createElement("input");
        submitBtn.setAttribute("type", "submit");
        submitBtn.setAttribute("value", "Submit");

        // Append labels and inputs to form container
        formContainer.appendChild(userNameLabel);
        formContainer.appendChild(userNameInput);
        formContainer.appendChild(document.createElement("br"));
        formContainer.appendChild(passwordLabel);
        formContainer.appendChild(passwordInput);
        formContainer.appendChild(document.createElement("br"));
        formContainer.appendChild(submitBtn);

        formContainer.addEventListener("submit", validate);
    }

    function validate(event){
        // Function to validate username (alpha-numeric only)
        function isUserName() {
            const txtUName = document.getElementById("uname");
            const letters = /^[A-Za-z0-9]+$/;
            if (!txtUName.value.match(letters)) {
                alert('User name must contain only alphanumeric characters!');
```

```
                return false;
            }
            return true;
        }

        // Function to validate password (at least 6 characters)
        function isPassword() {
            const txtPwd = document.getElementById("pwd");
            if (txtPwd.value.length < 6) {
                alert('Password length must be at least 6');
                txtPwd.value = "";
                txtPwd.focus();
                return false;
            }
            return true;
        }

        if(!isUserName() || !isPassword()){
            // Prevent form submission
            event.preventDefault();
        }
    }
</script>
```