Understanding the HTML DOM

Chirag Dhamija

August 27, 2025

What is HTML?

- **HTML** stands for **H**yper**T**ext **M**arkup **L**anguage.
- It is the standard markup language for creating web pages and web applications.
- HTML describes the structure of a webpage semantically and originally included cues for the appearance of the document.
- It is a cornerstone technology of the World Wide Web, alongside CSS and JavaScript.

Basic Structure of an HTML File

- An HTML document has a nested structure defined by tags.
- The basic skeleton includes:
  - `<!DOCTYPE html>` declaration.
  - `<html>` element that wraps the entire content.
  - `<head>` section for metadata and links to scripts/styles.
  - `<body>` section for the visible content.

Common HTML Tags

- **Headings**: `<h1>` to `<h6>` define headings of different levels.
- **Paragraph**: `<p>` defines a paragraph.
- **Div**: `<div>` is a block-level container for grouping content.
- **Span**: `<span>` is an inline container for grouping content.
- **Anchor**: `<a>` defines a hyperlink.
- **Image**: `<img>` embeds an image.
- **Unordered List**: `<ul>` creates a bulleted list.
- **List Item**: `<li>` defines an item in a list.
- **Script**: `<script>` embeds JavaScript code.
- **Link**: `<link>` links external resources like CSS.

HTML Tags Examples

Anchor Tag Example

```
<a href="https://www.example.com">Visit Example.com</a
    >
```

- Creates a clickable link that navigates to the specified URL.

Image Tag Example

```
<img src="image.jpg" alt="Description of image">
```

- Embeds an image into the webpage.
- `src` specifies the image source.
- `alt` provides alternative text for accessibility.

JavaScript

- JavaScript is a versatile, high-level programming language that powers modern web development.
- It enables dynamic and interactive behavior in web applications, such as form validation, animations, and real-time updates.
- Introduced by Netscape in 1995, it has evolved into one of the most widely used languages today.
- Key Uses:
  - Frontend: Manipulates the DOM (Document Object Model) to create interactive UIs.
  - Backend: Runs on Node.js for server-side applications.
  - Mobile Apps: Frameworks like React Native allow cross-platform mobile app development.
  - Game Development: Libraries like Three.js enable 3D graphics and game creation.

Variables and Data Types

- Variables are containers for storing data values.
- Declaration Keywords:
  - `var`: Legacy keyword with function scope (avoid in modern code).
  - `let`: Block-scoped variable (preferred for mutable values).
  - `const`: Block-scoped constant (immutable value after declaration).
- Data Types:
  - **Primitive**: Basic types stored directly in memory.
    - `String`, `Number`, `Boolean`, `Undefined`, `Null`, `Symbol`, `BigInt`.
  - **Non-Primitive**: Complex types stored as references.
    - `Objects`, `Arrays`, `Functions`.

Operators in JavaScript

- Operators perform operations on variables and values.
- Categories:
  - **Arithmetic**: Perform math operations (`+`, `-`, `*`, `/`, `%`).
  - **Comparison**: Compare values (`==`, `===`, `!=`, `!==`, `>`, `<`, `>=`, `<=`).
  - **Logical**: Combine conditions (`&&`, `||`, `!`).
  - **Assignment**: Assign values (`=`, `+=`, `-=`, `*=`, `/=`).
- **Type Coercion vs Strict Equality**:
  - `==`: Compares values after type coercion.
  - `===`: Compares both value and type (strict equality).

Examples

```
console.log(10 + 5);          // 15
console.log(10 == "10");      // true (type coercion)
console.log(10 === "10");     // false (strict equality)
```

Control Flow

- Control flow determines the order in which code is executed.
- Conditional Statements:
  - `if-else`: Executes code based on a condition.
  - `switch`: Matches a value against multiple cases.
- Loops:
  - `for`: Iterates a fixed number of times.
  - `while`: Repeats while a condition is true.
  - `do-while`: Executes at least once before checking the condition.
  - `for...of`: Iterates over iterable objects (arrays, strings).
  - `for...in`: Iterates over object properties.

```
// if-else example
let age = 20;
if (age > 18) {
    console.log("Adult");
} else {
    console.log("Minor");
}
```

```
// for loop example
for (let i = 0; i < 5; i++) {
    console.log("Iteration: ", i);
}

// while loop example
let count = 0;
while (count < 3) {
    console.log("Count: ", count);
    count++;
}
```

```
// do-while loop example
let num = 5;
do {
    console.log("Number is: ", num);
    num--;
} while (num > 0);

// for...of loop example
let fruits = ["Apple", "Banana", "Cherry"];
for (let fruit of fruits) {
    console.log("Fruit: ", fruit);
}

// for...in loop example
let person = {name: "Alice", age: 22, city: "New York"};
for (let key in person) {
    console.log(key + ": " + person[key]);
}
```

What is the DOM?

- The Document Object Model (DOM) is a programming interface for web documents.
- It represents HTML or XML documents as a tree of objects.
- It defines:
  - HTML elements as objects.
  - Properties, methods, and events for all HTML elements.

Key Features

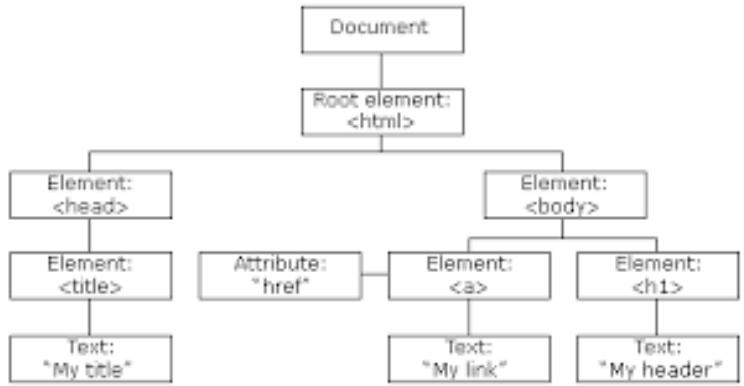
- Platform and language-independent.
- Enables dynamic manipulation of content, structure, and style.

The HTML DOM and JavaScript

- The HTML DOM is an API (Application Programming Interface) for JavaScript.
- With JavaScript, you can:
  - Add, change, or remove HTML elements and attributes.
  - Add, change, or remove CSS styles dynamically.
  - React to HTML events.
- When a web page is loaded, the browser creates a tree-like structure (DOM tree) to represent it.

DOM Tree Structure

- The DOM represents the document as a hierarchical tree structure.
- Nodes in the tree represent elements, attributes, and text.
- Node relationships include:
  - Parent, child, and sibling nodes.



Accessing the DOM

- To work with HTML elements, you need to find them first.
- Common methods for accessing elements:
  - `document.getElementById()`
  - `document.getElementsByTagName()`
  - `document.getElementsByClassName()`
  - `document.querySelector()`
  - `document.querySelectorAll()`

Code Example

```
var myElement = document.getElementById("intro");
console.log(myElement.textContent);
```

Finding HTML Elements

- Methods to locate elements in the DOM:
  - By ID: `document.getElementById("id")`
  - By tag name: `document.getElementsByTagName("tag")`
  - By class name: `document.getElementsByClassName("class")`
  - By CSS selector: `document.querySelector("selector")`
- Examples:
  - `var x = document.getElementsByTagName("p")`
  - `var x = document.querySelectorAll(".intro")`

DOM Manipulation - Basics

- The Document Object Model (DOM) represents the structure of an HTML document as a tree of nodes.
- Selecting Elements:
  - `document.getElementById(id)`: Selects an element by its ID.
  - `document.querySelector(selector)`: Selects the first matching element.
  - `document.querySelectorAll(selector)`: Selects all matching elements as a NodeList.
- Modifying Elements:
  - Change content: `element.innerHTML = "New Content"`.
  - Change styles: `element.style.color = "red"`.

Manipulating the DOM

- You can dynamically modify content, attributes, and styles.
- Example:

Code Example

```
1 const newElement = document.createElement('div');
2 newElement.textContent = 'Hello, DOM!';
3 document.body.appendChild(newElement);
```

DOM Manipulation - Examples

```
// Selecting and Modifying Elements
document.getElementById("demo").innerHTML = "Hello, World!";
document.querySelector("h1").style.color = "blue";

// Creating and Appending Elements
let newDiv = document.createElement("div");
newDiv.innerHTML = "This is a new div";
document.body.appendChild(newDiv);

// Modifying Attributes
let img = document.querySelector("img");
img.setAttribute("src", "new-image.jpg");
img.setAttribute("alt", "A beautiful landscape");
```

DOM Manipulation - Examples

```
// Adding and Removing Classes
let button = document.querySelector("button");
button.classList.add("btn-primary");
button.classList.remove("btn-disabled");

// Traversing the DOM
let listItem = document.querySelector("li");
console.log(listItem.parentElement); // <ul>
console.log(listItem.nextElementSibling); // Next <li>
console.log(listItem.previousElementSibling); // Previous <li>

// Removing an Element
let oldElement = document.getElementById("old");
oldElement.remove();
```

Events in the DOM

- Events are actions (e.g., clicks, hovers) occurring in the browser.
- Event listeners allow you to respond to these actions.

Code Example

```
const button = document.getElementById('myButton');
button.addEventListener('click', () => {
    alert('Button clicked!');
});
```

DOM Manipulation - Events

- Events allow interaction with users (e.g., clicks, mouse movements, keypresses).
- Adding Event Listeners:
  - Use `addEventListener` to attach events to elements.
  - Syntax: `element.addEventListener(event, handler)`.
- Common Events:
  - `click`: Triggered when an element is clicked.
  - `mouseover`: Triggered when the mouse hovers over an element.
  - `keydown`: Triggered when a key is pressed.
  - `submit`: Triggered when a form is submitted.

```
// Button Click Event
document.getElementById("btn").addEventListener("click", () =>
    alert("Button Clicked!");
});

// Mouseover Event
document.getElementById("box").addEventListener("mouseover",
    console.log("Mouse hovered over the box");
});

// Keydown Event
document.addEventListener("keydown", (event) => {
    console.log("Key pressed: ", event.key);
});

// Form Submit Event
document.getElementById("myForm").addEventListener("submit",
    event.preventDefault();
    console.log("Form submitted!");
```

Form Validation

- Validate input fields dynamically on submission.
- Display error messages below invalid fields.

```
function validateForm() {
    let name = document.getElementById("name").value;
    if (!name) {
        document.getElementById("nameError").innerText = "Name
        return false;
    }
    alert("Form submitted successfully!");
    return true;
}
```

Best Practices & Debugging

- Follow best practices to write clean and maintainable code:
  - Use `const` and `let` instead of `var`.
  - Prefer strict equality (===) over loose equality (==).
  - Handle errors gracefully using `try-catch`.
- Debugging Tools:
  - Browser DevTools: Inspect elements, log messages, and debug code.
  - Console Methods: `console.log`, `console.error`, `console.warn`.
  - Breakpoints: Pause execution to inspect variables and step through code.

```
try {
    JSON.parse("invalid JSON");
} catch (error) {
    console.error("Error parsing JSON:", error);
}
```

Conclusion

- JavaScript is essential for modern web development.
- Key Takeaways:
  - Understand the basics (variables, operators, control flow).
  - Master DOM manipulation and event handling for interactive UIs.