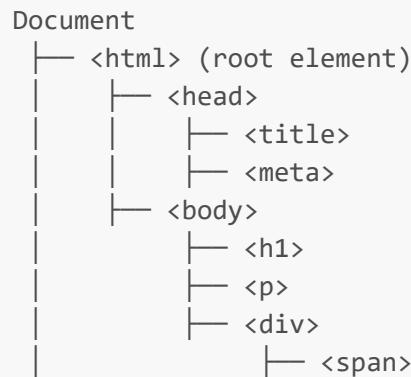


Document Object Model (DOM) - Study Notes

Introduction to DOM

- The Document Object Model (DOM) is a programming interface for web documents.
- It represents the HTML or XML document as a tree of objects.
- The DOM allows programs and scripts to dynamically access and update the content, structure, and style of a webpage.

Structure of the DOM



- The root node is `<html>`, containing `<head>` and `<body>`.
- Each element, attribute, and text inside HTML is a node in the DOM tree.

Accessing DOM Elements

- JavaScript provides several methods to access elements in the DOM:
 - `document.getElementById(id)`: Selects the element with the specified ID. This is the fastest and most efficient method.
 - `document.getElementsByClassName(className)`: Selects all elements with the specified class name. Returns an HTMLCollection (an array-like object).
 - `document.getElementsByTagName(tagName)`: Selects all elements with the specified tag name. Returns an HTMLCollection.
 - `document.querySelector(selector)`: Selects the first element that matches a CSS selector (e.g., #myId, .myClass, p).
 - `document.querySelectorAll(selector)`: Selects all elements that match a CSS selector. Returns a NodeList (similar to an HTMLCollection).

Manipulating DOM Elements

- Changing Content:
 - `element.innerHTML = "New content";`
 - `element.textContent = "New text";`
- Changing Attributes:

- `element.setAttribute(attributeName, value);`
- `element.getAttribute(attributeName);`
- `element.removeAttribute(attributeName);`
- Changing Styles:
 - `element.style.property = "value";`
- Creating Elements:
 - `document.createElement(tagName)`: Creates a new element node
- Adding Elements:
 - `parentNode.appendChild(childNode);`: Appends a child node to a parent node
 - `parentNode.insertBefore(newNode, existingNode);`: Inserts a new node before an existing node.
- Removing Elements:
 - `element.remove();`

Events

Web pages are interactive. Users click buttons, hover over things, type in text boxes, and so on. `addEventListener` lets you write JavaScript code that reacts to these user actions (or other events). Without it, your web pages would be static and boring.

- Syntax:

```
element.addEventListener(eventType, functionToExecute, useCapture);
```

```
<button id="myButton">Click me!</button>

<p id="myParagraph"></p>

<script>
const myButton = document.getElementById("myButton");
const myParagraph = document.getElementById("myParagraph");

myButton.addEventListener("click", function() {
    myParagraph.textContent = "You clicked the button!";
});
</script>
```

- Another Example:

```
<div id="myDiv" style="width: 100px; height: 50px; background-color: lightblue;">Hover over me!</div>

<script>
const myDiv = document.getElementById("myDiv");

myDiv.addEventListener("mouseover", function() {
```

```
myDiv.style.backgroundColor = "yellow";
});

myDiv.addEventListener("mouseout", function() {
  myDiv.style.backgroundColor = "lightblue";
});
</script>
```