DOM Manipulation

Inleiding

Stel je voor dat een webpagina een grote tekening is met allerlei onderdelen, zoals koppen, tekst en plaatjes. De **DOM** (Document Object Model) is als een soort "instructieboek" dat uitlegt hoe die tekening in stukjes is verdeeld, zodat we precies weten waar elk onderdeel zit.

Waarom is dit handig met JavaScript?

- **Veranderen van de tekening:** JavaScript is als een magische potlood dat de tekening kan aanpassen. Met de DOM kan JavaScript precies aanwijzen welk onderdeel van de tekening hij wil veranderen, toevoegen of verplaatsen, zonder dat je de hele tekening opnieuw moet maken.
- **Reageren op wat jij doet:** Stel je voor dat je op een knop klikt. JavaScript kan dat opvangen via de DOM en dan iets leuks doen, zoals een verborgen bericht laten zien of een plaatje veranderen.

Kortom, de DOM zorgt ervoor dat JavaScript precies weet hoe de pagina in elkaar zit, zodat je webpagina interactief en dynamisch kan worden.

Theorie

(English below)

DOM is het Document Object Model, dat is zeg maar de hele HTML pagina. Je kunt met JavaScript de DOM anapassen. Je kunt bijvoorbeeld een kleur van een element aanpassen, of de inhoud van een element aanpassen.

Dit kan natuurlijk ook met PHP, maar daarvoor moet je (terug) naar de server. Met JS kan je lokaal (front End) dingen aanapssen. Dat is vaak sneller en ziet er vaak beter uit.

Om de DOM aan te passen, heb je een paar specifieke commando's. De belangrijkste worden hier besproken.

1. Toegang krijgen tot Elementen

Om de inhoud te manipuleren, moet je eerst toegang krijgen tot de elementen. Veelgebruikte methoden om toegang te krijgen tot DOM-elementen zijn:

- `document.getElementById(id)`: Selecteert een enkel element op basis van zijn ID.
- `document.getElementsByTagName(name)`: Geeft een verzameling van elementen terug met de gegeven tag-naam.
- `document.getElementsByClassName(name)`: Geeft een verzameling van elementen terug die de gegeven klassenaam hebben.
- `document.querySelector(selector)`: Geeft het eerste element terug dat overeenkomt met een gespecificeerde CSS-selector.
- `document.querySelectorAll(selector)`: Geeft een lijst terug van alle elementen die overeenkomen met een gespecificeerde CSS-selector.

2. Wijzigen van Elementeigenschappen

Zodra je een referentie naar een element hebt, kun je de eigenschappen ervan wijzigen. Dit omvat het wijzigen van de innerlijke HTML, tekstinhoud, attributen en stijl van het element. Bijvoorbeeld:

- `element.innerHTML`: Haalt op of stelt de HTML- of XML-markering in die binnen het element is opgenomen.
- `element.textContent`: Stelt de tekstuele inhoud van een element en zijn nakomelingen in of retourneert deze.
- `element.setAttribute(name, value)`: Voegt een gespecificeerd attribuut toe aan een element, of werkt de waarde ervan bij als het al bestaat.
- `element.style.property`: Wijzigt de stijl van een element, waarbij property een CSS-eigenschap is (in camelCase, bijvoorbeeld `backgroundColor`).

3. Creëren en Verwijderen van Elementen

JavaScript stelt je in staat om nieuwe elementen te creëren en deze in het DOM in te voegen of bestaande elementen te verwijderen:

- `document.createElement(tagName)`: Creëert een nieuw element met de gegeven tag-naam.

- `parentNode.appendChild(childNode)`: Voegt een knooppunt toe als het laatste kind van een ouder knooppunt.
- `parentNode.insertBefore(newNode, referenceNode)`: Voegt een nieuw knooppunt in vóór het referentieknooppunt als een kind van een gespecificeerd ouderknooppunt.
- `parentNode.removeChild(child)`: Verwijdert een kindknooppunt uit het DOM.
- `element.remove()`: Verwijdert het element uit het DOM.

4. Event Handling

Het toevoegen van event listeners aan elementen stelt je in staat om code uit te voeren als reactie op gebruikersacties, zoals klikken, toetsenbordinvoer of muisbewegingen:

- `element.addEventListener(event, function, useCapture)`: Voegt een event handler toe aan een element. Veelvoorkomende events zijn `click`, `mouseover`, `mouseout`, `keydown`, `keyup`, enz.

5. Doorlopen en Wijzigen

Het DOM biedt methoden en eigenschappen om knooppunten te doorlopen en te wijzigen:

- `element.childNodes`: Geeft een live NodeList van kindknooppunten van een element terug.
- `element.parentElement`: Geeft het ouder-element van het gespecificeerde element terug.
- `element.nextSibling`: Geeft de volgende sibling in de boom terug.
- `element.previousSibling`: Geeft de vorige sibling in de boom terug.

Theory in English

(Dutch above)

The DOM is the Document Object Model, which is essentially the entire HTML page. You can modify the DOM using JavaScript. For example, you can change the color of an element, or modify the content of an element.

This can also be done with PHP, but that requires a (return) trip to the server. With JS, you can make local (front-end) adjustments. This is often faster and usually looks better.

To modify the DOM, you need a few specific commands. The most important ones are discussed here.

1. Accessing Elements

To manipulate the content, you first need to access the elements. Common methods to access DOM elements include:

- `document.getElementById(id)`: Selects a single element by its ID.
- `document.getElementsByTagName(name)`: Returns a collection of elements with the given tag name.
- `document.getElementsByClassName(name)`: Returns a collection of elements that have the given class name.
- `document.querySelector(selector)`: Returns the first element that matches a specified CSS selector.
- `document.querySelectorAll(selector)`: Returns a list of all elements that match a specified CSS selector.

2. Changing Element Properties

Once you have a reference to an element, you can modify its properties. This includes changing the element's inner HTML, text content, attributes, and style. For example:

- `element.innerHTML`: Gets or sets the HTML or XML markup contained within the element.
- `element.textContent`: Sets or returns the textual content of an element and its descendants.
- `element.setAttribute(name, value)`: Adds a specified attribute to an element, or updates its value if it already exists.
- `element.style.property`: Changes the style of an element, where `property` is a CSS property (in camelCase, e.g., `backgroundColor`).

3. Creating and Removing Elements

JavaScript allows you to create new elements and insert them into the DOM or remove existing elements:

- `document.createElement(tagName)`: Creates a new element with the given tag name.
- `parentNode.appendChild(childNode)`: Appends a node as the last child of a parent node.
- `parentNode.insertBefore(newNode, referenceNode)`: Inserts a new node before the reference node as a child of a specified parent node.
- `parentNode.removeChild(child)`: Removes a child node from the DOM.
- `element.remove()`: Removes the element from the DOM.

4. Event Handling

Adding event listeners to elements enables you to execute code in response to user actions, such as clicks, keyboard input, or mouse movements:

- `element.addEventListener(event, function, useCapture)`: Attaches an event handler to an element. Common events include `click`, `mouseover`, `mouseout`, `keydown`, `keyup`, etc.

5. Traversal and Modification

The DOM provides methods and properties to traverse and modify nodes:

- `element.childNodes`: Returns a live NodeList of child nodes of an element.
- `element.parentElement`: Returns the parent element of the specified element.
- `element.nextSibling`: Returns the next sibling in the tree.
- `element.previousSibling`: Returns the previous sibling in the tree.

Revision #6
Created 4 April 2024 21:53:44 by Max
Updated 26 February 2025 09:19:38 by Max