EPICODE

JavaScript 2

Giorno 2

EPICODE

DOM Traversing & DOM Manipulation

DOM Traversing & DOM Manipulation

Il DOM Traversing & Manipulation è la possibilità di utilizzare JavaScript per selezionare e manipolare strutture del Document Object Model (DOM)

La selezione si serve di specifici metodi che intercettano l'elemento tramite:

- Id
- Classe
- Nome del tag
- Selettori CSS

Questi sono solo i principali metodi applicati all'oggetto document, la cui lista è particolarmente nutrita.

La manipolazione degli elementi avviene attraverso l'assegnazione di metodi all'elemento selezionato.

Struttura html

```
<div id="test"></div>
```

Selezione in base all'id con il metodo getElementById()

```
document.getElementById("test");
```

Manipolazione di una proprietà (innerHTML)

```
document.getElementById("test").innerHTML =
"<span>nuovo elemento</span>";
```

Metodi e proprietà per gli elementi HTML

Qualsiasi tipo di elemento HTML può essere selezionato e manipolato in JS.

Proprietà e metodi di JS possono essere applicati ad ogni elemento HTML tramite il corrispondente nodo del DOM.

Esempi

creiamo un nuovo elemento lista
document.createElement(“*li*”);

Selezioniamo elementi secondo una classe html
document.getElementsByClassName(“*nomeClasse*”);

Selezioniamo un elemento tramite l’id e aggiungiamo una classe
document.getElementById(“*nomeID* ”).className = “nomeClasse”;

Selezioniamo un elemento con una classe
document.querySelector(“*.nomeClasse*”);

Selezioniamo tutti gli elementi con una stessa classe
document.querySelectorAll(“*.nomeClasse*”);

JavaScript attraverso la proprietà style permette la manipolazione dello stile di un elemento HTML selezionato

document.getElementById(“nomeId”).style;

All’oggetto style può essere assegnata una proprietà CSS per una formattazione specifica

document.getElementById(“nomeId”).style.color = “green”;

Esempi

document.querySelector(“*.nomeClasse*”).style.color = “green”;

document.getElementById(“nomeId”).style.fontSize = “large”;

È anche possibile creare nuovi elementi da inserire nella pagina per la creazione di contenuti dinamici. Non siamo quindi limitati alla sola selezione di nodi esistenti.

Basterà utilizzare il metodo createElement specificando il tipo di elemento che si vorrà creare

document.createElement(“div”)

Esempi

```
const newDiv = document.createElement(“div”)
// elemento div creato e accessibile tramite variabile newDiv
```

Una volta creato il nuovo nodo questo esisterà solo in memoria, e potremo modificarlo per contenuto e per stile prima di *inserirlo effettivamente* nella pagina.

Esempi

```
const newDiv = document.createElement(“div”)

newDiv.innerText = “nuovo contenuto testuale”
newDiv.style.backgroundColor = “red”
```

N.B. Ora come ora il nuovo div non fa ancora parte del DOM pertanto non lo vedremo da nessuna parte nella pagina web

L'ultimo passaggio che rimane da fare è quindi quello di inserire il nuovo nodo creato nella pagina. Il punto preciso dipende dal caso specifico e dalla struttura che vorremo creare.

Per inserire un elemento nella pagina bisogna selezionare il nodo del genitore in cui inserire il nuovo nodo e usare il metodo .appendChild()

body.appendChild(newDiv)

questo farà sì che il nuovo elemento non esisterà più solo in memoria ma farà parte del nuovo DOM tree

Esempi

```
const body = document.querySelector("body")
const newDiv = document.createElement("div")

newDiv.innerText = "nuovo contenuto testuale"
newDiv.style.backgroundColor = "red"

body.appendChild(newDiv)
```

Navigare fra gli elementi del DOM

Partendo dalla definizione e dalla natura del DOM, ogni suo elemento è un cosiddetto **nodo** che è attraversabile, raggiungibile tramite metodi in JavaScript.

I nodi sono organizzati gerarchicamente e sono in relazione l'uno con l'altro: elemento parent, child, sibling, ancestor (proprio come in CSS).

Due sono i nodi radice con cui è possibile accedere a tutto il documento:

document.body
document.documentElement

Queste proprietà esistenti su ogni nodo selezionato, permettono di navigare fra i nodi del DOM tree:

- parentNode
- closest()
- childNodes
- firstElementChild
- lastElementChild
- nextSibling
- previousSibling

```
elementNode.firstChild.innerText;
// innerText recupera il contenuto di un nodo testuale
```

EPICODE

GRAZIE

EPICODE