Corso Enaip
Rimini 10/03/2023

Agenda

- Document
- Introduzione agli Eventi

Document

Alberatura DOM

La struttura portante di un documento HTML è rappresentata dai tags.

Secondo il **Document Object Model** (DOM), ogni tag HTML **è un oggetto**. I tags annidati vengono chiamati "*figli*" del tag che li racchiude.

Tutti questi oggetti sono accessibili usando **JavaScript**.

Ad esempio, **document.body** è l'oggetto che rappresenta il tag **<body>**.

Alberatura DOM

```
document.body.style.background = 'red'; // rende il background rosso

setTimeout(() => document.body.style.background = '', 5000); // reimposta il background dopo 5 secondi
```

Alberatura DOM

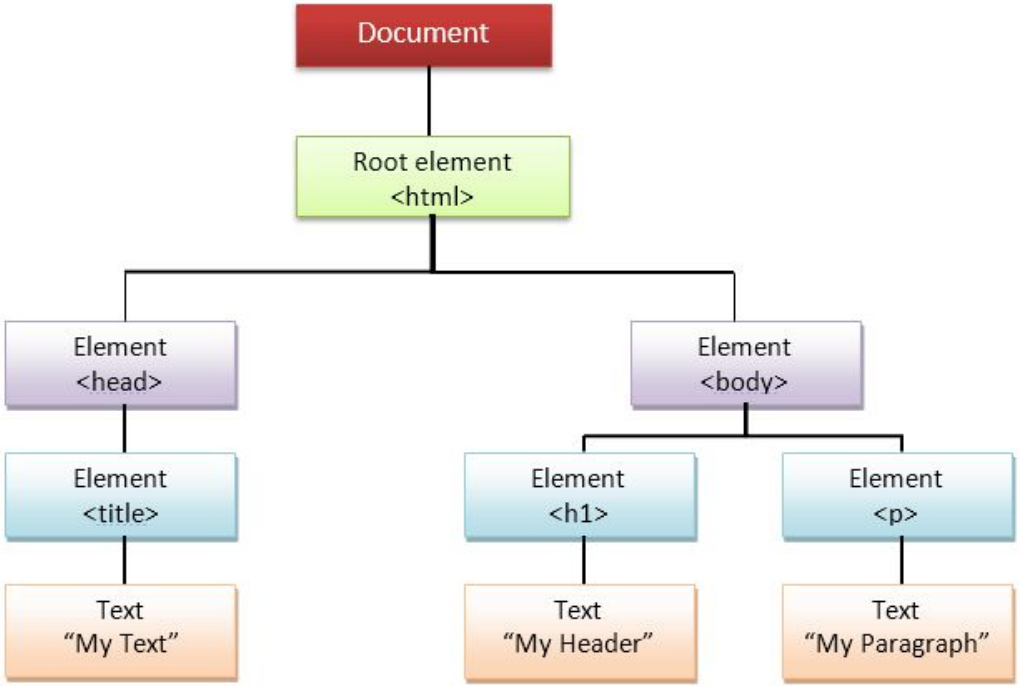
```
<html>
<head>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet"
        integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">
</head>
<body>
    <form>
        <div class="form-group">
            <label for="exampleInputEmail1">Email address</label>
            <input type="email" class="form-control" id="exampleInputEmail1" aria-describedby="emailHelp"
                placeholder="Enter email">
            <small id="emailHelp" class="form-text text-muted">We'll never share your email with anyone else.</small>
        </div>
        <div class="form-group">
            <label for="exampleInputPassword1">Password</label>
            <input type="password" class="form-control" id="exampleInputPassword1" placeholder="Password">
        </div>
        <div class="form-check">
            <input type="checkbox" class="form-check-input" id="exampleCheck1">
            <label class="form-check-label" for="exampleCheck1">Check me out</label>
        </div>
        <button type="submit" class="btn btn-primary">Submit</button>
    </form>
</body>
</html>
```

Percorrere il DOM

Il DOM ci consente di fare qualsiasi cosa con gli elementi ed il loro contenuto, ma prima dobbiamo essere in grado di raggiungere l'oggetto DOM corrispondente.

Tutte le operazioni sul DOM iniziano con l'oggetto **document**. Questo è il principale "punto d'ingresso" per il DOM, dal quale possiamo accedere a qualsiasi nodo.

Percorrere il DOM



Percorrere il DOM

I nodi in cima del documento HTML sono disponibili direttamente come proprietà di **document**:

<html> = document.documentElement

Il nodo del DOM più in alto è document.documentElement, esso corrisponde al tag <html>.

<body> = document.body

Un altro nodo DOM largamente utilizzato è l'elemento <body>, ossia document.body.

<head> = document.head

Il tag <head> è disponibile come document.head.

Percorrere il DOM - childNodes, firstChild, lastChild

- La proprietà **childNodes** raccoglie tutti i nodi figlio, inclusi quelli di testo.
- Le proprietà **firstChild** e **lastChild** permettono un accesso più veloce al primo ed all'ultimo figlio.

Percorrere il DOM - childNodes, firstChild, lastChild

```
<html>
<body>
    <div>Questo è un DIV</div>
    <ul>
        <li>Questo è un li</li>
    </ul>
    <div>Questo è un altro DIV</div>
    <script>
        for (var i = 0; i < document.body.childNodes.length; i++) {
            alert(document.body.childNodes[i]); // Text, DIV, Text, UL, ..., SCRIPT
        }
    </script>
</body>
</html>
```

Percorrere il DOM - nextSibling, previousSibling, parentNode

Il fratello successivo è nella proprietà **nextSibling**, quello precedente in **previousSibling**.

Il genitore è disponibile come **parentNode**.

Percorrere il DOM - nextSibling, previousSibling, parentNode

```
<html>
<body>
    <div>Questo è un DIV</div>
    <ul>
        <li>Questo è un li</li>
    </ul>
    <div>Questo è un altro DIV</div>
    <script>
        // il genitore di <body> è <html>
        alert( document.body.parentNode === document.documentElement ); // true

        // dopo di <head> c'è <body>
        alert( document.head.nextSibling ); // HTMLBodyElement

        // prima di <body> c'è <head>
        alert( document.body.previousSibling ); // HTMLHeadElement
    </script>
</body>
</html>
```

Percorrere il DOM - Solo gli elementi

Le proprietà di navigazione viste finora fanno riferimento a tutti i nodi. Ad esempio, in childNodes possiamo trovare: nodi elemento, nodi di testo, ed anche nodi commento se ce ne sono.

Ma per alcuni compiti non vogliamo nodi di testo o di commento. Vogliamo solo manipolare nodi che rappresentano i tags e che costituiscono la struttura della pagina.

- **children** – solo i figli che sono nodi elemento.
- **firstElementChild**, **lastElementChild** – il primo e l'ultimo elemento figlio.
- **previousElementSibling**, **nextElementSibling** – gli elementi vicini.
- **parentElement** – l'elemento genitore.

Percorrere il DOM - Solo gli elementi

```
<html>
<body>
    <div>Questo è un DIV</div>
    <ul>
        <li>Questo è un li</li>
    </ul>
    <div>Questo è un altro DIV</div>
    <script>
        for (var elem of document.body.children) {
            alert(elem); // DIV, UL, DIV, SCRIPT
        }
    </script>
</body>
</html>
```

Domande?

Ricerca: getElement*

Le proprietà di navigazione del DOM funzionano bene per gli elementi vicini. E quando non lo sono? Come possiamo ottenere un elemento arbitrario della pagina?

Ci sono altri metodi di ricerca per questo.

Ricerca: document.getElementById

```
<div id="elemento">
    <div id="elemento-content">Ciao</div>
</div>

<script>
    // trova l'elemento
    var elem = document.getElementById('elemento');

    // rendi il suo background rosso
    elem.style.background = 'red';
</script>
```

Ricerca: document.getElements*

- **elem.getElementsByTagName(tag)**
  - cerca gli elementi con il tag specificato e ritorna una loro collection. Il parametro tag può anche essere *, che equivale a “qualsiasi tag”
- **elem.getElementsByClassName(className)**
  - ritorna gli elementi con la data classe.
- **document.getElementsByName(name)**
  - ritorna gli elementi con l’attributo name, ovunque nel documento.

Ricerca attraverso un selettore CSS

- querySelectorAll
  - Tra i metodi più versatili, elem.querySelectorAll(css) ritorna tutti gli elementi contenuti in elem che combaciano con il selettore CSS specificato.
- querySelector
  - La chiamata a elem.querySelector(css) ritorna il primo elemento che combacia con il selettore CSS specificato.
- matches
  - Il metodo elem.matches(css) non cerca nulla; controlla semplicemente se elem combacia con il selettore CSS specificato, e ritorna true o false.
- closest
  - Il metodo elem.closest(css) cerca l'antenato più vicino che combacia il selettore CSS specificato. elem stesso è incluso nella ricerca.

Domande?

Le classi del DOM

EventTarget – è la classe radice (root class) “astratta”. Gli oggetti di questa classe non vengono mai creati. Serve solo come base, in questo modo tutti i nodi del DOM supportano i cosiddetti “eventi” che vedremo nelle slides successivamente.

Node – anche questa è una classe “astratta” che serve da base per i nodi del DOM. Fornisce le funzionalità principali della struttura gerarchica: parentNode, nextSibling, childNodes e così via (si tratta di getter). Dalla classe Node non vengono mai creati oggetti, tuttavia da questa ereditano classi corrispondenti a nodi concreti, nella fattispecie: Text per i nodi di testo, Element per i nodi elemento e quelli meno ricorrenti come Comment per i nodi commento.

Element – è la classe base per gli elementi del DOM. Fornisce le funzionalità di navigazione tra elementi come nextElementSibling, children ed i metodi di ricerca come getElementsByTagName, querySelector. Un browser non supporta solo HTML, ma anche XML e SVG. La classe Element serve da base per le classi più specifiche: SVGElement, XMLElement e HTMLElement.

HTMLElement – è, infine, la classe base per tutti gli elementi HTML. Essa è ereditata da elementi HTML concreti:

- **HTMLInputElement** – la classe per gli elementi <input>,
- **HTMLBodyElement** – la classe per gli elementi <body>,
- **HTMLAnchorElement** – la classe per gli elementi <a>,

…e così via, ogni tag ha una propria classe che espone proprietà e metodi specifici.

Le classi del DOM

https://dom.spec.whatwg.org/

innerHTML

La proprietà innerHTML consente di ottenere una stringa contenente l'HTML dentro l'elemento.

Possiamo anche modificarla e pertanto è uno dei più potenti strumenti per cambiare l'HTML di un elemento della pagina.

innerHTML

```
<html>

    <body>
        <p>A paragraph</p>
        <div>A div</div>

        <script>
          alert( document.body.innerHTML ); // legge il contenuto corrente
          document.body.innerHTML = 'The new BODY!'; // lo rimpiazza
        </script>

      </body>

</html>
```

outerHTML

La proprietà outerHTML contiene tutto l’HTML di un elemento. In pratica equivale a innerHTML più l’elemento stesso.

outerHTML

```
<html>

    <body>
        <div>Hello, world!</div>

        <script>
          var div = document.querySelector('div');

          // sostituisce div.outerHTML con <p>...</p>
          div.outerHTML = '<p>A new element</p>'; // (*)

          // Wow! 'div' non è cambiato!
          alert(div.outerHTML); // <div>Hello, world!</div> (**)
        </script>

      </body>

</html>
```

outerHTML

È molto semplice commettere un errore a questo punto: modificare div.outerHTML e procedere con div come se avesse recepito il nuovo contenuto. Ma questo non avviene. Tale convinzione è corretta per **innerHTML**, ma non per **outerHTML**.

Possiamo scrivere tramite elem.outerHTML, ma dovremmo tenere bene presente **che non cambia l'elemento** ('elem') su cui stiamo scrivendo, sostituisce invece il nuovo HTML al suo posto. Per avere un **riferimento valido** al nuovo elemento dobbiamo interrogare nuovamente il DOM.

Domande?

Altre proprietà

- hidden
  - L’attributo “hidden” e la corrispettiva proprietà del DOM specificano se l’elemento debba essere visibile o meno.
- value
  - il valore di <input>, <select> e <textarea> (HTMLInputElement, HTMLSelectElement…).
- href
  - il valore dell’attributo “href” di <a href="..."> (HTMLAnchorElement).
- id
  - il valore dell’attributo “id” per tutti gli elementi (HTMLElement).

Gli attributi di un tag (o elemento)

Tutti gli attributi sono accessibili utilizzando i seguenti metodi:

- elem.hasAttribute(name) – controlla l’esistenza.
- elem.getAttribute(name) – ritorna il valore.
- elem.setAttribute(name, value) – imposta un valore.
- elem.removeAttribute(name) – rimuove l’attributo.

Gli attributi di un tag (o elemento)

```
<html>

    <body something="non-standard">
        <script>
            alert(document.body.getAttribute('something')); // non standard
        </script>
    </body>

</html>
```

Modificare il DOM

Per creare nodi DOM, ci sono due metodi:

- document.createElement(tag)
  - Crea un nuovo nodo elemento con il tag fornito in input
- document.createTextNode(text)
  - Crea un nuovo nodo di testo con il testo fornito

Modificare il DOM

```
<html>

<body something="non-standard">
    <script>
        // 1. Creare l'elemento <div>
        var div = document.createElement('div');

        // 2. Impostare la sua classe ad "alert"
        div.className = "alert";

        // 3. Riempirlo con il contenuto
        div.innerHTML = "<strong>Ciao!</strong> Forza Roma.";
    </script>
</body>

</html>
```

I metodi d’inserimento

Ecco i metodi d’inserimento; specificano i differenti posti dove inserire un elemento:

- node.append(...nodi o stringhe) – appende nodi o stringhe alla fine di node,
- node.prepend(...nodi o stringhe) – inserisce nodi o stringhe all’inizio di node,
- node.before(...nodi o stringhe) –- inserisce nodi o stringhe prima di node,
- node.after(...nodi o stringhe) –- inserisce nodi o stringhe dopo node,
- node.replaceWith(...nodi o stringhe) –- rimpiazza node con i nodi o le stringhe passate.

Rimozione di nodi

Per rimuovere un nodo, abbiamo a disposizione il metodo **node.remove()**.

document.write

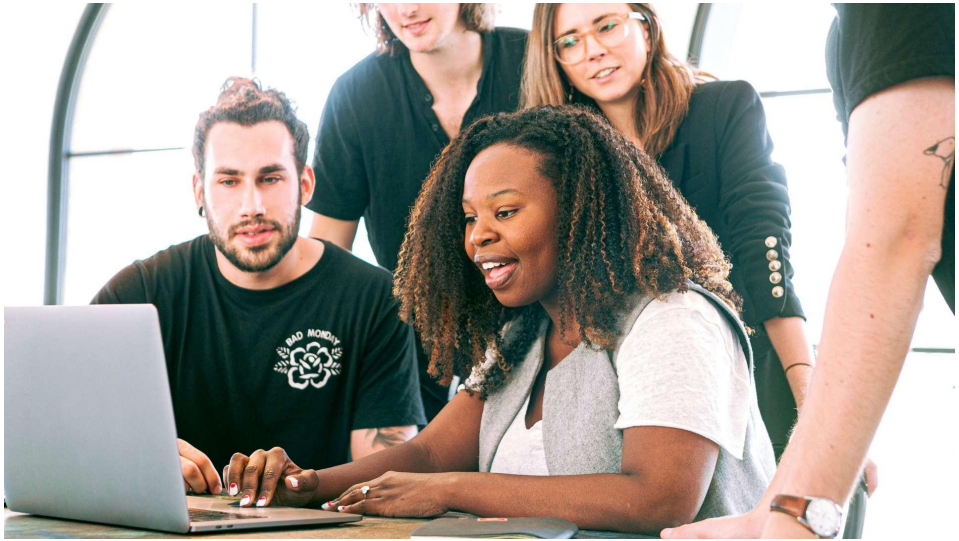
La chiamata a **document.write(html)** inserisce html nella pagina “all’istante”. La stringa html può essere generata dinamicamente, quindi in un certo senso è flessibile.

Negli script moderni possiamo trovarlo raramente a causa della seguente importante limitazione:

La chiamata a document.write funziona solo mentre la pagina sta caricando.

Esercitazione 1

Utilizzando le Classi Rubrica e Contatto dell’esercizio svolto la volta scorsa creiamo una pagina web che visualizzi la lista dei contatti presenti nella rubrica, la lista deve essere numerata e avere un colore di background diverso alternato.



Introduzione agli eventi

Gli eventi

Un evento è un segnale che sta ad indicare che è avvenuto qualcosa. Tutti i nodi DOM generano questi segnali (anche se gli eventi non sono limitati al DOM).

Eventi del mouse

- **click** – quando si clicca col mouse su un elemento (i dispositivi touch lo generano tramite il tocco).
- **contextmenu** – quando si clicca col tasto destro su un elemento.
- **mouseover** / **mouseout** – quando il cursore passa sopra/abbandona un elemento.
- **mousedown** / **mouseup** – quando viene premuto/rilasciato il pulsante del mouse su un elemento.
- **mousemove** – quando si sposta il mouse.

Eventi da tastiera

keydown e **keyup** – quando viene premuto e rilasciato un tasto.

Eventi degli elementi del form

submit – quando l’utente invia un <form>.

focus – quando l’utente attiva il focus su un elemento, ad esempio su un <input>.

Eventi del Document

DOMContentLoaded – quando l'HTML viene caricato e processato, e la costruzione del DOM è stata completata.

Eventi dei CSS

transitionend – quando termina un’animazione CSS (CSS-animation).

Event Handler

Per reagire agli eventi possiamo assegnare un gestore (handler). Questo non è altro che una funzione che viene eseguita contestualmente alla generazione di un evento.

I gestori, quindi, sono un modo per eseguire codice JavaScript al verificarsi delle azioni dell'utente ed esistono vari modi per assegnare un evento.

Event Handler

```
<html>

<body something="non-standard">
    <input value="Cliccami" onclick="alert('Click!')" type="button">
    <input id="elem" type="button" value="Cliccami">
    <script>
        var elem = document.getElementById("elem");
        elem.onclick = function() {
            alert('Grazie');
        };
    </script>
</body>

</html>
```

Domande?

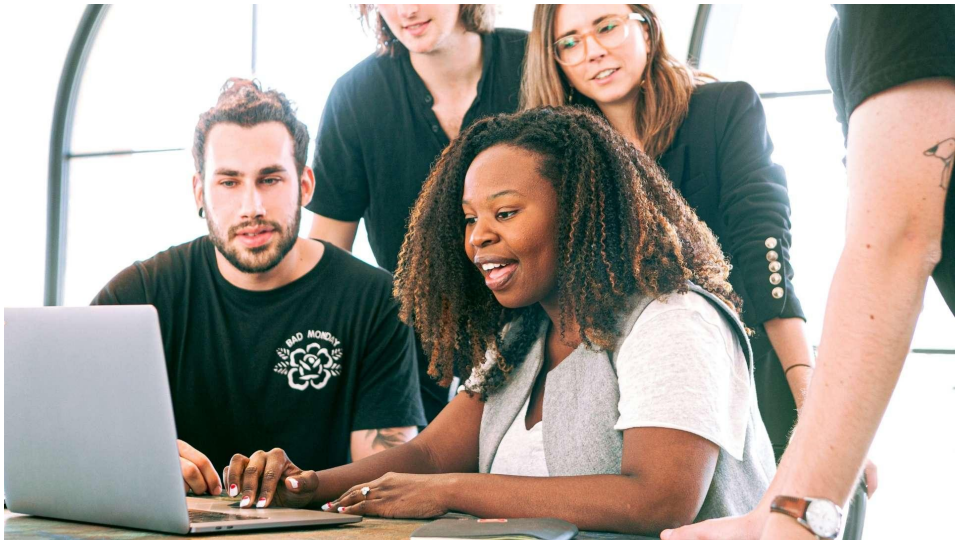
Esercitazione 2

Utilizzando le Classi Rubrica e Contatto dell’esercizio svolto la volta scorsa creiamo una pagina web che visualizzi la lista dei contatti, numerandoli.

Un form dove poter inserire un nuovo contatto. Un pulsante su ogni riga per rimuovere il contatto.

Un pulsante per cancellare tutti gli elementi della lista contatti.

Un form per effettuare la ricerca.



Prossima lezione

- Gli eventi
- Gestione degli errori
- Promises, async/await