Document Object Model

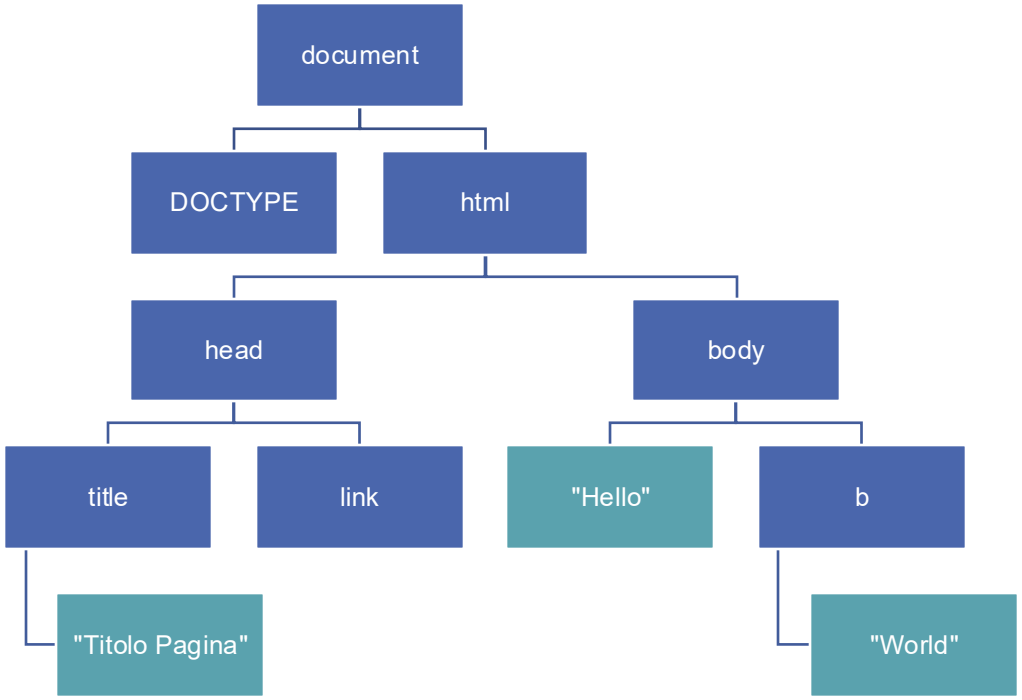
DOM

Il Document Object Model è una rappresentazione ad albero di oggetti della struttura di una pagina HTML.

Gli oggetti sono utilizzabili dal codice javascript tramite metodi/proprietà che questi espongono.

La pagina HTML è rappresentata nel DOM come una struttura ad albero, dove ogni nodo rappresenta un tag HTML o un contenuto testuale di un nodo HTML.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Titolo Pagina</title>
    <link rel="stylesheet" href="./style.css">
  </head>
  <body>
    Hello <b>World</b>
  </body>
</html>
```



DOM

Ogni nodo del DOM è un oggetto che implementa interfacce che dipendono dalla sua tipologia.

Tutti i nodi implementano le interfacce generiche **Node** e **Element**, che forniscono metodi e proprietà di base, e interfacce specifiche quali **HTMLHtmlElement**, **HTMLParagraphElement**, **HTMLDivElement** … che forniscono metodi e proprietà associate alla particolare tipologia di nodo.

L'elenco completo delle interfacce e delle relative proprietà/metodi è disponibile nel sito:

https://developer.mozilla.org/en-US/docs/Web/API

DOM – oggetto document

L'oggetto document è implementa la classe Element ed è sempre presente in quanto rappresenta la radice del DOM.

Partendo da questo oggetto possiamo "navigare" la struttura dell'albero utilizzando i metodi disponibili.

```
console.log(document); //restituisce l'HTML della pagina

console.dir(document); //restituisce la le proprietà e gli oggetti che compongono document
```

DOM – Selezionare elementi

L'oggetto document mette a disposizione una serie di metodi tramite i quali navigare nel DOM e selezionarne porzioni:

```
Element getElementById(elementId);
```

Seleziona l'elemento che ha l'attributo id uguale a quello passato come parametro. Restituisce null in caso non esista. L'elemento restituito implementa a sua volta l'interfaccia Element.

Nel caso di più elementi con lo stesso id i browser tipicamente restituiscono il primo. Ma non è un comportamento standard in quanto L'ATTRIBUTO ID DEVE ESSERE UNIVOCO.

DOM – Selezionare elementi

```
NodeList getElementsByName(elementName);
```

Seleziona tutti gli elementi che hanno l'attributo name uguale a quello passato come parametro. Restituisce null in caso non ne esista nessuno.

Il risultato è una collezione di elementi che implementano l'interfaccia Element.

```
let elements = document.getElementsByName('username');

elements.forEach(item => console.log(item) );
```

DOM – Selezionare elementi

Anche l'interfaccia Element, implementata anche da document, mette a disposizione una serie di metodi tramite i quali navigare nel DOM e selezionarne porzioni.

In questo caso la ricerca avviene solamente nel sottoalbero che ha come radice l'elemento su cui è stato chiamato il medoto.

```
NodeList getElementsByTagName(tagName);
```

Seleziona tutti gli elementi realizzati con il tag uguale a quello passato come parametro. Restituisce null in caso non ne esista nessuno.

Il risultato è una collezione di elementi che implementano l'interfaccia Element.

```
var forms = document.getElementsByTagName('form'); //cerca in tutto il documento html

var inputs = forms[0].getElementsByTagName(input'); //cerca nella prima form
```

DOM – Selezionare elementi

```
NodeList getElementsByClassName(classNames);
```

Seleziona tutti gli elementi che hanno la classe passata come parametro. Restituisce null in caso non ne esista nessuno.

Il risultato è una collezione di elementi che implementano l'interfaccia Element.

La ricerca avviene solamente nel sottoalbero che ha come radice l'elemento su cui è stato chiamato il medoto.

Come parametro possono essere passati i nomi di più classi, separati da spazio. Verranno selezionati tutti gli elementi che contengono tutte le classi specificate, indipendentemente dall'ordine.

DOM – Selezionare elementi

```
NodeList querySelectorAll(selector);
```

Seleziona tutti gli elementi che soddisfano il selettore css passato come parametro. Restituisce null in caso non ne esista nessuno.

Il risultato è una collezione di elementi che implementano l'interfaccia Element.

La ricerca avviene solamente nel sottoalbero che ha come radice l'elemento su cui è stato chiamato il medoto.

Esiste anche:

```
Element querySelector(selector);
```

che restituisce il primo elemento che soddisfa il selettore css

DOM – Selezionare elementi

Dato un nodo che implementa la classe Element è possibile navigare nell'albero anche con alcune proprietà:

- `Element parentNode` - restituisce il nodo padre
- `NodeList childNodes` - restituisce una collezione con tutti i nodi figli (di tutti i tipi)
- `Node firstChild` - restituisce il primo figlio (di qualsiasi tipo)
- `Node lastChild` - restituisce l'ultimo figlio (di qualsiasi tipo)
- `Node nextSibling` - restituisce il fratello successivo (di qualsiasi tipo)
- `Node previousSibling` - restituisce il fratello precedente (di qualsiasi tipo)

DOM – Selezionare elementi

Molti metodi della slide precedente restituiscono i nodi figli di qualsiasi tipo. Molto spesso però cerchiamo solo i nodi di tipo Element. Esistono metodi per accedere solamente a questi nodi:

- `NodeList children` - restituisce una collezione con tutti i nodi figli di tipo Element
- `Element firstElementChild` - restituisce il primo figlio di tipo Element
- `Element lastElementChild` - restituisce l'ultimo figlio di tipo Element
- `Element nextElementSibling` - restituisce il fratello successivo di tipo Element
- `Element previousElementSibling` - restituisce il fratello precedente di tipo Element
- `int childElementCount` - restituisce il numero di figli di tipo Element

DOM – Modificare la struttura ad albero

L'oggetto document contiene metodi per creare nuovi elementi:

- `Element createElement(tagName)` - restituisce un oggetto Element relativo al tag passato come parametro
- `Text createTextNode(text)` - restituisce un oggetto text con il contenuto passato come parametro

I singoli nodi contengono metodi per clonare/aggiungere/rimuovere elementi:

- `Node appendChild(newChild)` – Accoda l'elemento passato come parametro ultimo figlio del chiamante.
- `Node insertBefore(newChild, refChild)` – Inserisce il nuovo elemento prima dell'elemento refChild. RefChild deve essere un figlio del chiamante.
- `Node removeChild(child)` – Rimuove dal DOM il figlio passato come parametro.
- `Node replaceChild(newChild, oldChild)` – sostituisce oldChild con newChild

DOM – Modificare il contenuto di un elemento

Gli Element hanno alcune proprietà che permettono di leggerne e modificarne il contenuto:

- `innerText` - restituisce tutto il testo, VISIBILE a video, presente all'interno di un elemento e dei suoi sottoelementi. Non vengono restituiti i tag html
- `innerHTML` - restituisce tutto l'html presente all'interno di un elemento e dei suoi sottoelementi. Vengono pertanto restituiti anche i tag html.
- `outerText` – identico a innerText se utilizzato in lettura. In scrittura però sostituisce anche l'elemento corrente.
- `outerHTML` - identico a innerHtml se utilizzato in lettura. In scrittura però sostituisce anche l'elemento corrente.
- `textContent` – analogo a innerText con la differenza che restituisce in contenuto anche di eventuali tag presenti nel documento ma non visualizzati a video

DOM – innerText vs textContent

```
<ul id="elenco">
	<li>Html</li>
	<li>Css</li>
	<li>Javascript</li>
	<li style="display:none">Typescript</li>
</ul>
```

- `document.getElementById("elenco").innerText` restituisce Html, Css, Javascript
- `document.getElementById("elenco").textContent` restituisce Html, Css, Javascript e Typescript

DOM – Modificare gli attributi di un elemento

Gli Element hanno alcuni metodi che permettono di accedere agli attributi:

- `getAttribute(name)` - restituisce il valore dell'attributo passato come parametro.
- `setAttribute(name, value)` – imposta l'attributo passato come primo parametro al valore passato come secondo parametro
- `removeAttribute(name)` – rimuove l'attributo passato come parametro
- `hasAttribute(name)` – verifica l'esistenza dell'attributo passato come parametro. Restituisce un boolean.

DOM – Modificare gli attributi di un elemento

Gli attributi sono anche mappati come proprietà dell'oggetto Element, utilizzando la seguente convenzione per i nomi:

- Se l'attributo è formato da una sola parola allora la proprietà ha lo stesso nome dell'attributo, tutto in minuscolo.
- Se l'attributo è formato da più parole (per esempio readonly), la proprietà utilizza la notazione camel case con la prima lettera in minuscolo (per esempio readOnly)
- Se l'attributo ha un nome uguale ad una parola chiave di javascript (per esempio for) allora la proprietà avrà il prefisso html e il nome dell'attributo sarà scritto con la prima lettera maiuscola (per esempio htmlFor)
- Il tipo di dato della proprietà riflette il tipo di dato del relativo attributo (per esempio: readOnly --> boolean)

DOM – Modificare gli attributi di un elemento

L'attributo class ha una gestione dedicata tramite due proprietà:

- className – Restituisce l'intero contenuto dell'attributo
- classList – Restituisce una collezione contenente tutte le classi. E' una proprietà in sola lettura e può essere modificata con i metodi add(nomeClasse), remove(nomeClasse) e toggle(nomeClasse)

DOM – Gestione degli eventi

La gestione degli eventi serve per eseguire codice javascript per nel momento in cui si verifica una determinata condizione. Per esempio:

- La pagina ha terminato il suo caricamento (onload)
- L'utente ha cliccato su un elemento (onclick)
- Il mouse si è soffermato su un elemento (onmouseover)
- Il mouse ha lasciato un elemento (onmouseout)
- E' stato premuto un pulsante (onkeydown)
- …

Gli eventi sono pertanto associati ad elementi. E' cioè l'elemento che definisce il codice javascript da eseguire a fronte di un determinato evento.

DOM – Gestione degli eventi

Esistono 3 modi per associare codice javascript ad un evento:

Direttamente nell'HTML:

```
<element eventName="javascript">
```

In javascript sfruttando il mapping attributi/proprietà:

```
element.eventName = function( ) {….}
```

In javascript sfruttando l'apposito metodo:

```
element.addEventListener("eventName", function( ) {….} )
```