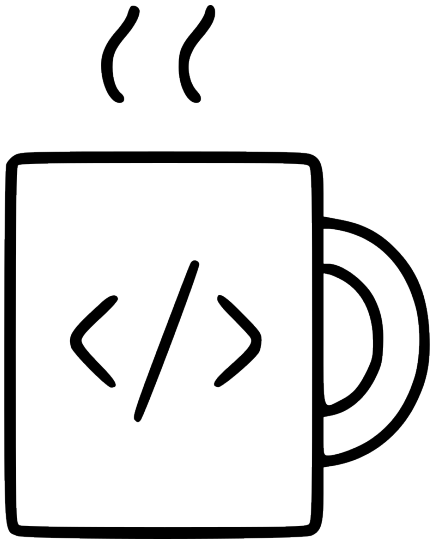
Umberto Emanuele

JavaScript 2
Giorno 5

Novembre 2023



PAUSA

Ci vediamo alle ore 11.00

Dom Traversing: gli eventi, le animazioni e i nodi

Gli eventi sul DOM

Una delle potenzialità maggiori di JS è quella di poter rispondere a degli eventi richiamati sugli elementi HTML.

Per esempio, il click su un bottone scatena un certo tipo di evento programmabile.

Tutti gli eventi del DOM derivano dall'oggetto Event.

Tipologie comuni di eventi

- Eventi del mouse: click, hover…
- Eventi della tastiera
- Caricamento di una pagina
- Caricamento di una immagine
- Eventi sul campo dell'input
- Invio di un form HTML

Esempio

Il metodo onclick può essere utilizzato per invocare una certa funzione che gestisce un evento.

```
<button onclick="funzione()">Clicca</button>
```

Mostra un testo nell'elemento selezionato.

La catena degli eventi

addEventListener

Scopo principale del metodo è assegnare ad un elemento selezionato un certo evento.

```
elemento.addEventListener(evento, funzione);
```

Possiamo però anche attribuire più eventi e funzioni al medesimo elemento.

```
document.getElementById("nomeID")
.addEventListener("click", nomeFunzione);
```

```
document.getElementTagName("button")
.addEventListener("click", nomeFunzione);

document.getElementTagName("button")
.addEventListener("onmouseover",
nomeFunzione1);

document.getElementTagName("button")
.addEventListener("onmouseout",
nomeFunzione2);
```

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 da JavaScript.

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

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

document.body
document.documentElement

JavaScript utilizza le seguenti proprietà per navigare fra i nodi del DOM:

- parentNode
- childNodes
- firstChild
- lastChild
- nextSibling
- previousSibling

```
elemento.childNodes[0].nodeValue;
```
nodeValue recupera il valore del nodo

Nodi e collezioni

I nodi possono essere agevolmente manipolati aggiungendone di nuovi o rimuovendo gli esistenti.

Una collezione (lista) di elementi può essere Generata usando il metodo getElementsByTagName che richiama ovviamente tutti gli elementi contrassegnati dal tag.

```
//Aggiungere un nodo
document.createElement("h1");
document.createTextNode(“Un nuovo titolo”);

//Rimuovere un nodo
elemento.remove();
elemento.childRemove(  elemento selezionato  );
```

```
document.getElementsByTagName(“p”);
```

Il metodo genera un oggetto che possiamo trattare in maniera analoga ad un array.

In questo caso abbiamo un array di tag paragrafo.

Form e validazione

Manipolazione degli elementi del form

JavaScript utilizza un oggetto Form per metodi e proprietà di gestione di elementi del form.

L'accesso agli elementi del form è possibile con la collection form o element del form

```
document.forms[0];
document.forms.item[1];
```

La collection element deve far riferimento ad un form specifico:

```
elemento("idForm").elements[1].value;
```

La proprietà length ci riporta il numero di form del documento oppure degli elementi in un form

I metodi item(index) e namedItem(id) riportano rispettivamente l'elemento secondo l'ordine dell' indice e dell'id

```
document.getElementById("idForm").elements.item(0).value;
```

```
document.getElementById("idForm").elements.namedItem("id").value;
```

All'oggetto Form possiamo attribuire due metodi principali: reset() e submit()

Le espressioni regolari

Le espressioni regolari (regex), sono sequenze di caratteri, numeri e simboli, scritti secondo regole standard, che disegnano dei pattern di riferimento.

Due sono gli usi delle regex:

- Creare un modello per validare in base al confronto con pattern uguali;
- Fornire un criterio per operazioni di ricerca e sostituzione

Una espressione regolare, oltre la sequenza alfanumeriche, compresi i caratteri speciali, utilizza dei metacaratteri.

Esempio:
la combinazione \s indica che il pattern deve considerare anche gli spazi.

Esempio di regex:

[bcd] => sono validi per il confronto o per la ricerca tutti i caratteri indicati fra le parentesi.

In merito ai form, le espressioni regolari sono particolarmente importanti quale forma di validazione degli input, specialmente nella creazione di pattern per la creazione di password o di nome identificativi.

Validazione dei form

La validazione dei campi di input di un form richiede sempre massima attenzione.

Naturalmente, vista la natura duplice del form come struttura di font end e di back end, parlando di validazione con JS, intendiamo una validazione lato client.

Tipologia di validazione:

- Campi vuoti
- Controllo di inserimento numerico o testuale
- Inserimento di sequenze numeriche di una determinata lunghezza
- Inserimento di sequenze con controllo di una espressione regolare come nelle password o nelle mail.

La validazione lato client controlla la correttezza dell'input prima che il dato sia inviato al server, quindi il controllo è a livello del browser.

API di validazione

Oltre le strutture di controllo condizionale con le quali possiamo validare un campo, JavaScript sfrutta le potenzialità di API dedicata con il metodi:

checkValidity() => controllo della validità dei dati

Per esempio si può controllare che un numero inserito in un input sia all'interno di un certo range.

Fra le proprietà vi è:

validationMessage

che ritorna un messaggio quando la condizione di validità è falsa

validity

con la quale è possibile settare diverse opzioni di controllo

Il controllo di validità fa largo uso degli attributi dei form propri di HTML5 come min e max

Esempio:
<input type="number" min="50" max="100">

Bom e Javascript

Con JavaScript possiamo manipolare e gestire dimensioni ed eventi legati alla finestra del browser.

Ecco i metodi utilizzabili con l'oggetto:

- window.open()
- window.close()
- window.moveTo()
- window.resizeTo()
- window.innerHeight
- window.innerWidth

Con JavaScript possiamo utilizzare 3 tipi di pop up gestiti dall'oggetto window:

- Pop up di alert => window.alert();
- Pop up di conferma => window.confirm();
- Pop up di prompt => window.prompt();

I metodi possono essere scritti omettendo window:

- alert('valore');
- confirm('valore');
- prompt('valore', 'testo di default');

Con JavaScript possiamo manipolare e gestire la storia legata agli eventi di navigazione.

L'oggetto è:

window.history

con i metodi:

history.back();
history.forward();

che replicano i relativi tasti del browser.

Con l'oggetto Navigator possiamo ricavare alcune informazioni di navigazione dell'utente relative al browser utilizzato.

L'oggetto window.navigator ha una discreta lista di metodi fra cui i principali:

- navigator.appName
- navigator.appCodeName
- navigator.platform => sistema operativo
- navigator.product
- navigator.appVersion => versione del browser
- navigator.userAgent
- navigator.language
- navigator.online

L'oggetto window usa due metodi per il settaggio dei tempi di azione della finestra del browser:

setTimeout(*funzione* , 1000);
esegue una funzione dopo un certo tempo

setInterval(*funzione* , 1000);
ripete ciclicamente secondo il tempo impostato una funzione

Gli eventi dei due metodi possono essere cancellati con:

clearTimeout();
clearInterval();

I cookie sono piccole stringhe di codice depositate a livello client.

Tipicamente sono impostati con una scadenza.

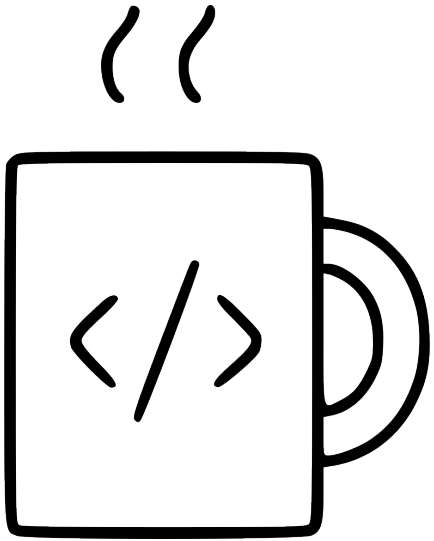
In Javascript si possono creare, modificare e cancellare i cookie raccogliendo informazioni di navigazione. Creare un cookie:

document.cookie = "nomeUtente = Mario Rossi";
Impostare un tempo:

"nomeUtente = Mario Rossi; expires=Thu, 26 october 2013 12:00:00 UTC";

Il cookie impostato può essere letto passandone il valore ad una variabile:

var mioCookie = document.cookie;



PAUSA

Ci vediamo alle ore 16.25

Introduzione a Ajax

Ajax è una tecnologia asincrona sviluppata in ambito JS ma adottabile con altri linguaggi di programmazione. L'acronimo sta per:

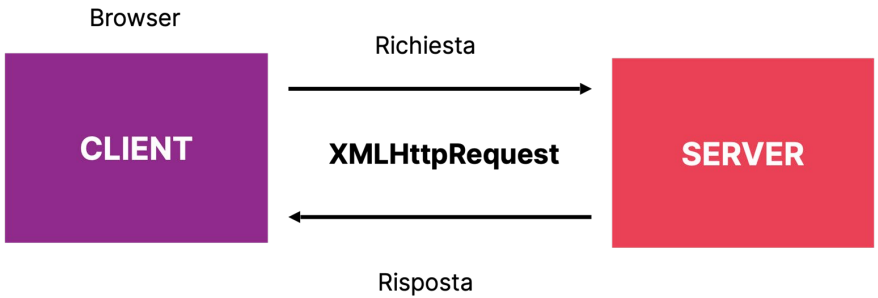
Asynchronous JavaScript And XML

Il punto di forza di questa tecnologia è che è possibile manipolare risorse senza reload della pagina, agendo su porzioni della stessa e richiamare o inviare dati ad un server dopo il caricamento della pagina.

Fondamentalmente Ajax utilizza l'oggetto del browser:

XMLHttpRequest

E gli oggetti relativi al DOM



Step del processo con l'oggetto XMLHttpRequest:

- Creazione di un'istanza
- Definire una funzione di callback
- Aprire una richiesta dell'oggetto
- Inviare la richiesta

```
Creazione:
var nuovaRichiesta = new XMLHttpRequest();

Callback:
xhttp.onload = funzione

Aprire  la richiesta:
xhttp.open("GET", risorsa);

Inviare la richiesta:
xhttp.send();
```

L'oggetto possiede diverse proprietà tra cui:

onreadystatechange => gestione del cambiamento degli stati
readyState => gestione degli stati
status => ritorna il numero standard del messaggio del server e del client

Gli stati della richiesta possono essere intercettati e gestiti con la proprietà readyState:

0: richiesta non inizializzata
1: connessione col server stabilita
2: richiesta ricevuta
3: richiesta in elaborazione
4: richiesta elaborata e risposta in arrivo

Gli esiti delle richieste possono essere gestite con la proprietà status.

Fondamentalmente si fa riferimento ai messaggi del client o del server, propri del protocollo HTTP ed espressi da un numero.

In particolare mettiamo in evidenza:

messaggi di successo: 200
Problemi del client, in particolare, risorsa non trovata: 400 oppure 403 per accesso non consentito
Problemi del server: 500

Una richiesta che raggiunga lo stato 4 e ritorna un messaggio 200, ha avuto esito positivo e riceverà la risposta desiderata.

Gestione del formato JSON

JSON cioè JavaScript Object Notation, è un diffusissimo metodo per la formattazione dei dati che devono essere ricevuti o inviati.

Come per Ajax, JSON è utilizzabile non solo nell'ambito di JS ma da tutti i linguaggi di programmazione.

JSON presenta i dati in forma di oggetto JavaScript.
Possiamo formattare qualsiasi tipo di dato:

- Stringa
- Numero
- Booleano
- Oggetti innestati
- Array

Anche i dati multimediali e i link possono essere inseriti nei dati.


```
[
  {
    "albumId": 1,
    "id": 1,
    "title": "accusamus beatae ad facilis cum similique qui sunt",
    "url": "https://via.placeholder.com/600/92c952",
    "thumbnailUrl": "https://via.placeholder.com/150/92c952"
  },
  {
    "albumId": 1,
    "id": 2,
    "title": "reprehenderit est deserunt velit ipsam",
    "url": "https://via.placeholder.com/600/771796",
    "thumbnailUrl": "https://via.placeholder.com/150/771796"
  },
  {
    "albumId": 1,
    "id": 3,
    "title": "officia porro iure quia iusto qui ipsa ut modi",
    "url": "https://via.placeholder.com/600/24f355",
    "thumbnailUrl": "https://via.placeholder.com/150/24f355"
  },
```

Gli oggetti JSON sono essenzialmente oggetti JS.

Queste sono le regole fondamentali:

- I dati sono organizzati in coppie nome/valore
- I dati sono separati da virgole
- Le parentesi graffe contengono gli oggetti
- Le parentesi quadre contengono gli array

L'elemento nome nella coppia nome/valore deve essere una stringa e contenuto in apici doppi:

"nome":"Mario",

Anche il valore richiede i doppi apici

```
}
"utente" = {
"nome":"Mario",
                "anni":52,
                "luogo":"Firenze",
                "cliente":false
},
}
```

I dati dei file json devono essere decodificati per poter essere correttamente manipolati nel codice.

Attraverso il metodo JSON.parse(); possiamo convertire i dati JSON in oggetti JavaScript

```
var oggJS = JSON.parse(
'{"nome":"Mario", “anni":52, “luogo”:"Firenze”}'
);
```

Il processo inverso è trasformare un oggetto JavaScript in dato formattato JSON.

Per questo viene utilizzato il metodo

JSON.stringify();

Esempio:

```
var oggJS = {nome: "Mario", anni: 30, luogo:
"New York”};
var oggJSON = JSON.stringify(oggJS);
```

- Le chiamate asincrone di Ajax possono richiamare dati da un file JSON.
- La tecnica permette di iniettare in strutture HTML dei dati asincroni da una sorgente JSON o viceversa.
- Lo schema permette di selezionare un elemento HTML.
- Esempio una tabella o una lista con select.
- I dati JSON dovranno essere decodificati / codificati.

Introduzione a ECMAScript

Come abbiamo già detto JS è stato elaborato ufficialmente nel 1995.
Al 1997 invece risale la sua standardizzazione in ECMA => European Computer Manufacturer's Association
A livello tecnico ECMA è uno standard per i linguaggi di scripting.
JavaScript è quindi una implementazione, probabilmente la più nota, di ECMAScript.

In sigla abbiamo la seguente notazione:

ES1, ES2…
A partire dal 2016 usiamo la notazione con l'indicazione dell'anno:
ECMAScript 2016
…
ECMAScript 2018 che è l'ultima

Rispetto alla scrittura vacilla di JS, ECMAScript presenta delle particolarità sintattiche e delle strutture di programmazione sue proprie che vengono normalmente adottate dai framework.

Una delle più ricche implementazioni del linguaggio è rappresentato dalla versione del 2015 di cui abbiamo già visto ampi esempi.

Per ricapitolare:

- Uso di let e const
- Funzione freccia
- Metodo Map sugli oggetti
- Evoluzione delle classi
- Promises
- Metodi Math
- Metodi degli array
- Ciclo for / of
- Funzioni con parametri di tipo rest
- Metodi delle stringhe startWith / endsWith
- Metodo Array.from

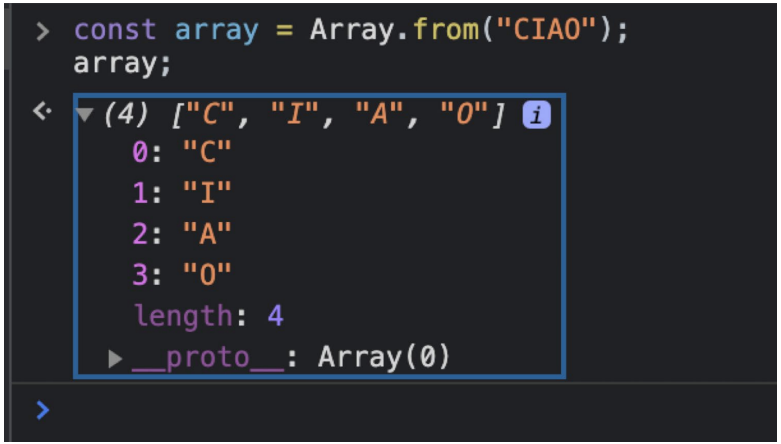
Due metodi:

string.startWith()
string.endsWith()

Permettono una selezione iniziale e finale di un valore indicato ritornando un valore booleano

Il metodo Array.from produce un oggetto a partire da elementi iterabili

```
let stringa = “Sto imparando JavaScript";

stringa.startsWith(“JavaScript”); => false

stringa.endsWith(“Javascript”); => true
```



In questo caso parliamo del rest parameter espresso dal simbolo (...) che permette di trattare un indefinito numero di parametri come un array.

```
function somma(...numeri) {
//istruzioni
 return somma;
}
```

```
> function somma(...numeri) {
    let somma = 0;
    for (let numero of numeri) somma += numero;
    return somma;
  }

  let x = somma(10, 21, 30, 7, 15);
  x;
< 83
>
```

Da una serie di oggetti possiamo creare una lista unica con Map() e usare i suoi metodi per la gestione dei dati.

```
const gatto = {nome: 'Gatto'};
const cane = {nome: 'Cane'};
const coniglio = {nome: 'Coniglio'};

const pet = new Map();
//configuriamo un elemento
pet.set(gatto, 'Ciro');
```

Il metodo get() ci permette di prelevare un valore:

```
pet.get(gatto);
```

Altri metodi ci permettono di cancellare tutti gli elementi clear() o uno specifico con delete().

Con il metodo has() indichiamo in booleano la presenza o no di una chiave.

L'operatore di spread permette una sorta di clone estendendo le proprietà di un oggetto all'altro.

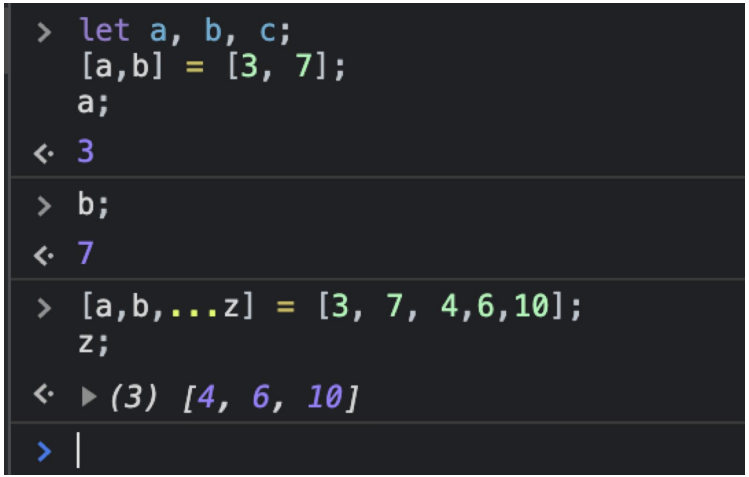
Si realizza con lo stesso simbolo dell'operatore rest (..proprietà).

```
const gatto = {
                zampe: 4,
                verso: 'miao'
                };

const cane = {
                ...gatto,
                verso: 'bau'
                };

                console.log(cane);
```

Questa è un'opzione introdotta con ECMAScript 2018.

Viene aggiunta una proprietà che ripulisce le risorse utilizzate in una promise che viene soddisfatta indipendentemente dal risultato.

La sintassi delle proprietà rest è un'espressione JavaScript che rende possibile decomprimere valori da array o proprietà da oggetti in variabili distinte.



challenge
NETWORK

shaping the skills of tomorrow

challengenetwork.it