- Alessandra Salvaggio, Gualtiero Testa -

JavaScript

Guida completa

Lavorare con le stringhe, i cicli, le funzioni, i form e gli eventi >>

Espressioni regolari, oggetti, array associativi, modalità strict >>

Funzioni avanzate, JSON, AJAX e REST >>

Canvas e geolocalizzazione >>

*pro DigitalLifeStyle

*pro
Digital*LifeStyle

JavaScript

Guida completa

Alessandra Salvaggio, Gualtiero Testa

EDIZIONI
LSWR

JavaScript | Guida completa

Autrice: Alessandra Salvaggio, Gualtiero Testa

Publisher: Marco Aleotti
Progetto grafico: Roberta Venturieri
Immagine di copertina: © TierneyMJ | Shutterstock

ISBN: 978-88-6895-631-8

EDIZIONI
LSWR

Via G. Spadolini, 7
20141 Milano (MI)
Tel. 02 881841
www.edizionilswr.it

Printed in Italy

Finito di stampare nel mese di marzo 2018 presso "Rotomail Italia" S.p.A., Vignate (MI)

(*) Edizioni Lswr è un marchio di La Tribuna Srl. La Tribuna Srl fa parte di LSWR GROUP.

Sommario

Introduzione

JavaScript non è certamente un linguaggio nuovo nel panorama della programmazione per il **web**, ma, negli **ultimi anni** sta riscontrando un rinnovato **interesse** e **successo**.

JavaScript esiste ormai da più vent'anni e ha vissuto, con alterne fortune, l'evoluzione del mondo Internet.
A volte snobbato e ritenuto un linguaggio più per amatori che per professionisti, a volte amato per la sua versatilità e semplicità, negli ultimi anni sta vivendo una vera e propria nuova giovinezza.
Reso standard e quindi più rigoroso da **IEEE**, il linguaggio inventato da **Brendan Eich** nell'ambito del progetto per il browser Netscape Navigator, è sempre più utilizzato per grossi progetti, non solo in ambito web. Esistono oggi intere applicazioni sviluppate in JavaScript.
Questo libro, per evidenti ragioni di spazio e per non disperdere troppo il discorso, si concentrerà essenzialmente sull'uso di JavaScript nel contesto web.
Si richiede ai lettori una conoscenza approfondita del linguaggio HTML (che viene dato per scontato) e conoscenze di CSS; non sono richieste, invece, particolari competenze nell'ambito della programmazione.
Il libro, infatti, parte dai concetti base fino ad affrontare argomenti più avanzati.
Il percorso è impegnativo ma, in queste pagine, cercheremo di guidare il lettore passo passo in modo che possa poi diventare autonomo e sviluppare propri progetti JavaScript.

Alla pagina http://www.sos-office.it/libri/javascript.html trovate tutti i file per svolgere gli esercizi del libro.

1

JavaScript

JavaScript è un linguaggio di programmazione nato oltre venti anni fa. Oggi vale ancora la pena di impararlo? Proviamo a rispondere alla domanda percorrendo anche l'evoluzione del linguaggio.

JavaScript (con le lettere J e S scritte in maiuscolo e spesso indicato con la sigla "JS") è un linguaggio di programmazione creato nel 1995 nell'ambito del progetto per il browser Netscape Navigator allo scopo di rendere la navigazione sul web più dinamica e interattiva.
Nella sua concezione iniziale, i programmi scritti in JavaScript vengono eseguiti dal browser web, quindi sul computer dell'utente che sta navigando e non sul server che ospita il sito.
Per le sue caratteristiche, JavaScript è un linguaggio un po' anomalo e non facilmente classificabile: il suo progettista **Brendan Eich** ha fatto scelte molto criticate dai puristi della programmazione, perché JavaScript riassume in sé, in modo non sempre armonico, caratteristiche di diversi linguaggi diffusi all'epoca della sua nascita.
Il risultato appare qualcosa di un po' "ibrido": per esempio, la sua sintassi deriva in gran parte da quella del linguaggio Java (per questo è stato chiamato JavaScript anche se la sintassi è l'unico collegamento tra i due linguaggi), ma JavaScript si discosta da Java anche in punti fondamentali, "filosofici". Un esempio su tutti: JavaScript definisce come Java (e altri linguaggi come SmallTalk) il concetto di oggetto (torneremo nel corso del libro su questo concetto), ma, per le sue caratteristiche, non può essere definito un vero linguaggio orientato agli oggetti.

JavaScript trae alcune caratteristiche dai linguaggi funzionali (come Scheme) e, infatti, definisce funzioni come elementi di prima classe (*first class function*), ma non è un puro linguaggio funzionale.
A questo "peccato originale", si sono aggiunti a rovinare la reputazione di JavaScript gli effetti della così detta "guerra dei browser", ossia una aspra competizione fra Microsoft e gli altri produttori di browser per accaparrarsi la leadership del mercato.
Cosa c'entra JavaScript con tutto questo? Cerchiamo di capire. Nel 1996 Microsoft ha creato, per Internet Explorer 3, il linguaggio jScript, una propria versione di JavaScript con caratteristiche specifiche e comportamenti diversi da JavaScript, il che rendeva siti sviluppati con una delle due versioni del linguaggio non compatibili con tutti i browser.
Tutti questi elementi di confusione, uniti alla concezione diffusa fra la fine degli anni Novanta e gli anni 2000 secondo cui le applicazioni "serie" girano sui server e non sul client (PC dell'utente), ha portato a una posizione molto negativa degli sviluppatori professionisti nei confronti di JavaScript, considerato solo un linguaggio per amatori e grafici.
Allora JavaScript è da considerare un linguaggio di serie B? Noi crediamo di no. Vediamo perché.

La rinascita di JavaScript

A partire dagli inizi di questo decennio, la situazione ha cominciato a cambiare. Sono nati tanti elementi nuovi che hanno contribuito a una diversa considerazione di JavaScript da parte della comunità degli sviluppatori; ne riassumiamo brevemente i principali.
Innanzi tutto la nascita di Chrome (2008), spinto dal gigante Google, porta a un forte cambiamento nel mercato dei browser: il suo motore di esecuzione di JavaScript ("V8") ha incrementato a tal punto la velocità di JavaScript da permettere di avere applicazioni complesse in esecuzione sul browser, senza cioè doverle prima installare sul PC. Questa possibilità oggi appare piuttosto normale, ma dieci anni fa non lo era affatto. Era una grande rivoluzione.

NOTA

Le statistiche di utilizzo dei browser (per esempio https://www.w3counter.com/globalstats.php) indicano che Chrome è il browser più utilizzato seguito a distanza da Safari (per il mobile). Internet Explorer, Opera e Firefox sono ampiamente indietro.

In secondo luogo, i nuovi processori multi-core (dual, quad...) permettono una vera esecuzione in parallelo dei programmi. Questa potenza può essere ben sfruttata da tecniche "asincrone" di programmazione che trovano in JavaScript un ambiente molto adatto (troverete i dettagli sull'esecuzione asincrona nel capitolo dedicato ad Ajax).
Non possiamo dimenticare la nascita del così detto web 2.0 che, con la sua forte integrazione tra i servizi offerti dai diversi siti (per esempio come i social si integrano tra di loro), ha spinto a creare applicazioni basate sui browser.
Non ultimo, la creazione di uno standard del linguaggio JavaScript accettato da tutti i produttori di browser ne ha portato a una sorta di "nobilitazione".
La creazione di uno standard, noto come ECMAScript, è avvenuta a opera di **ECMA** (European Computer Manufacturers Association, oggi nota come ECMA International). Questa associazione è stata fondata, con sede a Ginevra, nel 1961 con il compito di creare standard per il settore informatico e delle telecomunicazioni.
Nel 1996, Netscape ha affidato JavaScript ad ECMA con il preciso compito di creare uno standard. Nel 1997 è nata la prima edizione di ECMA-262, quello che oggi è noto genericamente come ECMAScript (nome di compromesso fra le istanze soprattutto di Netscape e Microsoft).
Da allora si sono succedute diverse versioni di ECMAScript di cui JavaScript è una implementazione.
Tutti questi avvenimenti hanno determinato un rinnovato e crescente interesse nei confronti di JavaScript, anche al di fuori dal suo ambito tradizionale (il web). Oggi, infatti, è possibile creare applicazioni stand alone in JavaScript che non necessitano di un browser. Fra queste applicazioni ricordiamo:

- applicazioni desktop (da installare sul PC) come Visual Code di cui parleremo in una appendice e che abbiamo usato per scrivere gli esempi di questo libro;
- applicazioni server grazie a piattaforme come Node.js;
- applicazioni mobile (tablet e smartphone);
- applicazioni IoT (Internet of Things, Internet delle Cose).

Queste applicazioni NON browser non fanno parte di questo libro che si concentrerà, invece, sull'utilizzo di JavaScript nell'ambito dei siti Internet e del web.
Questo non toglie, però, che l'uso di JavaScript al di fuori del web, abbia contribuito a farne crescere la forza e la popolarità.

Popolarità di JavaScript

Negli ultimi anni, come abbiamo avuto modo di vedere, JavaScript è diventato un linguaggio con un'importanza e una presenza significative, in molti ambiti e contesti diversi.

È sempre difficile misurare la popolarità e la diffusione di un linguaggio, ma i principali indici pongono JavaScript stabilmente tra i primi 10: per esempio, risulta al primo posto nella classifica di **GitHub** (https://octoverse.github.com/), al settimo in quella di **IEEE** (https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages), all'ottavo per **TIOBE** (https://www.tiobe.com/tiobe-index)...
Classifiche a parte, JavaScript è un linguaggio che vale la pena di considerare per i propri progetti e, se cercate una conferma in più, vi consigliamo una visita alla pagina http://shouldilearnjavascript.com/.

JavaScript e ECMAScript

Esistono otto versioni di ECMA-262; la pubblicazione di nuove versioni avviene ultimamente su base annuale, per cui, dall'edizione 6 dello standard, il nome della versione è derivato dall'anno di pubblicazione, anche se si è scelto di mantenere anche il numero d'ordine delle edizioni.
L'ultima versione attualmente disponibile è **ECMAScript 2017**, nota anche come **ES8** perché è l'ottava edizione dello standard. Una nuova versione, **ECMAScript 2018**, è già in fase avanzata di definizione.
Considerando l'ambito di nostro interesse (il web) quest'ultima edizione è fin troppo nuova per godere di un buon supporto da parte dei browser.
Possiamo dire che le versioni di riferimento sono la versione 5 e la versione 6 (la prima nota anche con il nome dell'anno, ECMAScript 2015).

NOTA

Il progetto Kangax su GitHub riporta alcune tabelle che dettagliano il livello di compatibilità dei browser (e di applicazioni come Node.js) in funzione della diversa edizione di ECMAScript:

ES5: https://kangax.github.io/compat-table/es5/
ES6: https://kangax.github.io/compat-table/es6/
ES2016/2017: http://kangax.github.io/compat-table/es2016plus/

Alla fine del 2017, tutti e quattro i principali browser (Chrome, Safari, Firefox e Microsoft Edge) hanno un supporto molto buono (> 95%) di **ES6** ed è a questa versione che faremo riferimento nel libro.

I fratelli di JavaScript

Le critiche dei teorici a JavaScript hanno favorito la nascita di varianti di JavaScript che lo "migliorano" nei punti considerati più deboli.

I programmi scritti con questi linguaggi, per non perdere la possibilità di essere usati all'interno dei browser, i quali supportano solo programmi scritti in JavaScript, devono essere poi tradotti in JavaScript. La traduzione è svolta in automatico da particolari programmi chiamati "transpiler" che prendono il programma scritto in un linguaggio e lo traducono in un programma scritto in un altro linguaggio.

Tra questi linguaggi varianti di JavaScript, uno dei più popolari è **TypeScript** di Microsoft. Angular, uno dei più utilizzati framework JavaScript, dalla sua versione 2, è scritto in TypeScript, non più in JavaScript.

2

Come scrivere il codice JavaScript

Prima di cominciare a descrivere le potenzialità di JavaScript, è bene fare una panoramica degli strumenti che possono essere utili al lavoro e mostrare come integrare i comandi di questo linguaggio nelle pagine HTML.

Argomenti trattati

- Strumenti di lavoro
- Integrazione di codice JavaScript nelle pagine HTML

Cominciamo dagli strumenti di lavoro.

Strumenti di lavoro

Sebbene, in linea teorica, per scrivere codice JavaScript e HTML sia sufficiente disporre di un editor di testi (come Blocco Note) e di un browser, quando si comincia a scrivere codice in modo un po' più "serio", gli strumenti giusti possono fare davvero la differenza. Senza pretesa di essere esaustivi, nelle pagine che seguono proponiamo una panoramica degli strumenti (gratuiti) più diffusi.

Editor di testo

Una prima alternativa a Blocco Note è costituita degli editor di testo "più evoluti".
Ne ricordiamo tre: **Notepad ++**, **Atom** e **Visual Studio**.
Il primo di questi, **Notepad ++** è un editor gratuito che può essere scaricato dal sito https://notepad-plus-plus.org/.

La sua interfaccia è semplice e di facile utilizzo.
Dispone di una serie di caratteristiche davvero utili:

- evidenziazione della sintassi;
- raggruppamento di porzioni omogenee di codice (*Syntax Folding*) in modo da poter nascondere o visualizzare porzioni di un documento lungo;
- evidenziazione della sintassi e *Syntax Folding* personalizzato dall'utente;
- evidenziazione delle parentesi;
- ricerca/sostituisci mediante espressioni regolari (Perl Compatible Regular Expression) ;
- autocompletamento della sintassi;
- segnalibri;
- visualizzazione a schede;
- visualizzazione di documenti affiancati per il confronto.

Atom è un editor gratuito scaricabile dal sito https://atom.io/ disponibile per più piattaforme (OS X, Windows e Linux). Può essere completato con diversi pacchetti open source e dispone di supporto al sistema di controllo di versione Git.
Fra i punti di forza di Atom ci sono:

- autocompletamento;
- evidenziazione della sintassi;
- funzionalità di ricerca e sostituzione fra più file;
- possibilità di aprire diversi file in pannelli affiancati per poterli confrontare.

Visual Studio Code è l'editor che abbiamo usato per scrivere gli esempi di questo libro. Abbiamo anche dedicato una appendice alla sua installazione e configurazione. È un editor sviluppato da Microsoft per più piattaforme (OS X, Windows e Linux). Si tratta di uno strumento gratuito scaricabile dalla pagina https://code.visualstudio.com/.
Dispone di Git integrato ed è integrabile con ulteriori pacchetti.
Fra i suoi punti di forza ci sono:

- autocompletamento;
- evidenziazione della sintassi;
- funzionalità di ricerca e sostituzione;
- possibilità di impostare breakpoint;
- lavora direttamente con file e cartelle senza la necessità di creare progetti.

Linter

Un **linter** è un programma che in genere si integra con un editor di codice e permette di evidenziare gli errori di sintassi o in generale di scrittura del codice.

Attualmente, uno dei linter per JavaScript più diffusi è ESLint (https://eslint.org/). Nella appendice dedicata a **Visual Studio Code** spiegheremo come integrare questo strumento utilissimo nell'editor Microsoft.

Web server

Molti degli esempi del libro possono essere eseguiti da file system, in alcuni casi, però, è necessario eseguire i test del codice da web server.
Potete utilizzare un servizio online oppure, ed è la soluzione che vi consigliamo per evitare di continuare a trasferire file via FTP (si rischia di testare i file non nell'ultima versione e fare confusioni), è possibile installare un web server sul proprio computer locale. Vi consigliamo **XAMPP** (https://www.apachefriends.org/it/index.html) alla cui installazione abbiamo dedicato una breve appendice.

Integrare i comandi JavaScript nelle pagine HTML

Dopo la panoramica sugli strumenti che possono facilitarvi il lavoro è bene vedere come far convivere JavaScript e HTML.
Sostanzialmente abbiamo due possibili soluzioni:

- inserire il codice JavaScript all'interno dello stesso file che contiene il codice HTML;
- scrivere il codice JavaScript in un file esterno con estensione .js e quindi richiamare questo file nel file HTML.

Cominciamo con un esempio del primo approccio:

```
<!doctype html>
<html lang="it">
<head>
  <meta charset="utf-8">
  <title>Prova1</title>
  <meta name="description" content="Prova1">
</head>
<body>
  <p id="output" />
  <script type="text/javascript">
    const msgHello = 'Buongiorno mondo';
    document.getElementById('output').innerHTML = msgHello;
  </script>
</body>
</html>
```

Indipendentemente da quello che fa questo codice e dalla sua sintassi, notate che le istruzioni JavaScript sono inserite nel tag `<script>` e viene specificato come attributo il tipo di codice contenuto nel tag (che è `text/javascript`).

Notate anche il codice è scritto in fondo alla pagina, appena prima della chiusura del tag `<body>`, in modo che il codice sia richiamato quando tutto il DOM del documento HTML è stato caricato.
Questa posizione del codice non è obbligatoria, ma caldamente consigliata in modo da evitare problemi dovuti al (temporaneo) mancato caricamento di elementi a cui il codice potrebbe fare riferimento.

NOTA

DOM, Document Object Model, è l'insieme degli oggetti di un documento HTML che possono essere manipolati via codice. Ne parleremo in dettaglio in un capitolo dedicato.

Il secondo approccio, che è più pratico se il codice JavaScript è complesso e lungo o da riutilizzare in più file, consiste nel creare due file distinti, il file HTML e il file .js con il codice JavaScript, e quindi richiamare il file .js nel file HTML come nell'esempio che segue:

```
<!doctype html>
<html lang="it">
<head>
  <meta charset="utf-8">
  <title>Prova1</title>
  <meta name="description" content="Prova1">
  <!--<link rel="stylesheet" href="css/styles.css?v=1.0">-->
</head>
<body>
  <p id="output" />
  <script type="text/javascript" src="codice.js"></script>
</body>
</html>
```

Il file .js viene richiamato mediante l'attributo `src` del tag `<script>`. Anche in questo caso, il codice JavaScript viene richiamato alla fine del corpo del documento.
In realtà i due approcci (file esterno, codice interno al file) possono coesistere. Magari nel file esterno c'è del codice generico che vale per più pagine, mentre all'interno del file HTML si tiene solo il codice specifico per la pagina o per un'azione puntuale. In questo caso, sono necessari due (o più) tag `<script>`, uno per richiamare il file esterno e uno per il codice gestito internamente.

```
<script type="text/Javascript" src="funzCookie.js"></script>
<script>
    let nomeUtente = leggiCookie('userName');
    if (nomeUtente != '') {
        document.getElementById('saluto').innerHTML = `Ciao ${nomeUtente}`;
    } else {
        nomeUtente = prompt('Non ti conoscono. Scrivi il tuo nome:', '');
        if (nomeUtente != '' && nomeUtente != null) {
            impostaCookie('userName', nomeUtente, 3);
        }
    }
</script>
```

Avvertenza

La soluzione più comune per la scrittura del codice è quella che prevede di tenere il codice in un file esterno da richiamare nel file HTML ma, per facilitare l'uso degli esempi del libro, abbiamo tenuto il codice interno al file HTML.
In questo modo, ogni file è "autosufficiente" e funzionante senza necessità di molte dipendenze.