Come utilizzare le espressioni regolari in JavaScript?

Dopo aver esaminato le basi teoriche e la sintassi delle espressioni regolari, **è importante sapere come utilizzare queste conoscenze in ambito produttivo**. Infatti, sebbene le regole siano comuni, ogni linguaggio di programmazione utilizza delle routine proprie: in questo articolo presenteremo l'implementazione delle regex in JavaScript ed i metodi principali associati alla loro gestione.

Le espressioni regolari in JavaScript vengono create come semplici stringhe, delimitate da *forward slash*, seguiti da alcuni caratteri speciali, detti flags, che modificano la metodologia di ricerca. Potremmo avere quindi una variabile (o più precisamente *un letterale*):

Il modello può essere una qualsiasi espressione regolare, mentre per i flag possono essere utilizzati i tre caratteri **g**, **i** o **m**:

- **g**: il modello viene cercato in tutta la stringa, invece di fermarsi dopo la prima occorrenza trovata.
- **i**: la ricerca non tiene conto della differenza di caratteri maiuscoli/minuscoli.
- **m**: la ricerca viene effettuata anche dopo il termine della prima riga di testo.

Un primo semplice esempio di espressione regolare potrebbe essere quello per la ricerca di una stringa:

Cerca nell'ipotetico testo la stringa "Your Inspiration Web", senza tener conto di eventuali differenze tra maiuscole e minuscole. Oppure potremmo riutilizzare l'espressione regolare utilizzata

per verificare una data:

Nell'implementazione delle espressioni regolari in JavaScript ci sono, però, alcune variazioni: quella del metacarattere 'punto', ovverosia "ogni carattere", è forse la più rilevante. Infatti è il carattere 'a capo' non viene corrisposto quando è settato il flag m . Per ovviare a questa mancanza si può utilizzare l'insieme di una classe di caratteri e della sua negazione, spesso nella forma "[\s\S]". Ma cosa significa? Controllando la tabella del precedente articolo puoi notare che la classe '\s' si riferisce a "tutti i caratteri di spaziatura". Aggiungendo nell'insieme '\S' che sta per "tutti i caratteri che **non sono spazi**" in pratica stiamo dicendo *"prendi gli spazi e tutti i caratteri che non sono spazi"*; in poche parole *"tutti i caratteri"*.

Quella del punto è l'unica assenza degna di nota per un uso 'normale' delle regex in Javascript (ci sono altre differenze). La domanda da porsi adesso è: come testare le nostre espressioni regolari in JavaScript?

L'oggetto RegExp

In realtà in JavaScript tutte le espressioni regolari (anche quelle di esempio realizzate sopra) **sono un'istanza dell'oggetto RegExp**. Non è rilevante a questo scopo conoscere la programmazione ad oggetti: basta solo tenere a mente che un oggetto è una struttura (come un array) che possiede delle proprietà (gli attributi) e dei comportamenti (i metodi). I metodi e gli attributi sono propri dell'oggetto a cui si riferiscono e la sintassi per richiamarli è la seguente:

Su un oggetto di tipo RegExp i metodi rilevanti sono due: **test ed exec**. Il primo esegue un test, appunto, di un'espressione regolare su una stringa, verificando che in essa vi sia almeno una corrispondenza. Il secondo invece, restituisce le informazioni sull'eventuale occorrenza trovata nella stringa.

Per quanto riguarda gli attributi sono degni di nota **lastIndex**, che contiene la posizione del prossimo carattere da esaminare da parte della regex, e **source**, che contiene la stringa dell'espressione regolare vera e propria.

Prima di provare queste funzioni, creiamo una pagina html per i test:

Scriveremo il nostro codice JavaScript all'interno del tag "script" in basso alla pagina.

Il metodo test()

Il metodo *test* accetta come parametro una stringa a cui applicare l'espressione regolare, e ha

come valore di ritorno un booleano: true, se la stringa corrisponde all'espressione regolare, false altrimenti.

Come primo esempio per il metodo *test*, cerchiamo in una stringa le corrispondenze della regex /CaSa/i:

Analizziamo passo per passo questa porzione di codice. Abbiamo:

1. creato la nostra espressione regolare, in questo caso una semplice stringa, impostando il flag *case-insensitive*;
2. inizializzato la stringa che fungerà da testo di ricerca, nella realtà questa sarà rappresentata dai valori di un form, oppure dal testo preso da una pagina web;
3. utilizzato il metodo test nel suo ambiente più congeniale, un costrutto *if*, se vi è una corrispondenza esegue l'azione relativa, altrimenti procede con il codice successivo.

Nota come, dopo aver creato l'espressione regolare, abbiamo richiamato il metodo "test" utilizzando la *dot-notation*: richiamare "test" senza anteporre il nome dell'oggetto risulterà un errore.

Il metodo exec

Sfruttando la stessa espressione regolare e la stessa stringa, vediamo come lavora il metodo "exec". Quest'ultimo prende come parametro una stringa a cui applicare la regex, ma ha come valore di ritorno un array contenente l'occorrenza trovata. Tale array possiede un attributo, **index** che contiene la posizione dell'occorrenza all'interno della stringa (il numero del carattere dall'inizio della stringa, più precisamente).

Vediamo un semplice esempio:

La creazione dell'espressione regolare e della stringa è identica a quella precedente. La prima differenza è il valore di ritorno del metodo, che viene salvato nella variabile "risultato". Questa variabile è un array contenente nella prima posizione l'occorrenza dell'espressione regolare trovata nella stringa, mentre nell'attributo "index" è memorizzata la posizione dell'occorrenza della stringa.

Una cosa interessante da notare è che, ripetendo l'esecuzione del metodo "exec" senza specificare il flag "g", si ottiene sempre e solo la prima occorrenza:

Per consentire a JavaScript di cercare le occorrenze successive della regex si deve, come accennato, utilizzare il flag "global" nella definizione dell'espressione regolare:

Specificando il flag global informiamo JavaScript che deve continuare la ricerca nel testo (un po' come la funzione "Trova successivo" degli editor di testo): quando non vi sono più occorrenze il metodo "exec" ritornerà il valore "NULL", che dovremo gestire adeguatamente.

Considerazioni finali

Le espressioni regolari possono semplificare molto la gestione di un'applicazione web. Va detto però che il motore delle regex richiede una percentuale di CPU molto elevata, quindi è bene utilizzarle con accortenza, verificando che non ci siano altri modi più veloci per ottenere lo stesso risultato. Ad esempio, volendo cercare tutti le righe di una stringa che terminino per punto e virgola, si potrebbe essere tentati di utilizzare l'espressione regolare:

Questa semplice espressione regolare potrebbe impegnare il tuo browser per svariati secondi (anche minuti, in caso di testi molto lunghi): questo perché **la ricerca non viene effettuata solo alla fine della stringa**, bensì comincia dal primo carattere, esaminandone uno alla volta, fino alla fine, per poi dare il risultato. Ma come migliorare questa ricerca?

Le espressioni regolari sono utili quando non si hanno ipotesi sul testo: in questo caso, invece, noi vogliamo sapere solo **se l'ultimo carattere è un punto e virgola**. Perché allora non effettuare il test solo su quell'ultimo carattere?

Come hai visto ho utilizzato il metodo *charAt* per recuperare l'ultimo carattere e confrontarlo con la mia stringa di ricerca.

In generale dunque, quando si ha a che fare con la ricerca o la sostituzione di semplici stringhe, potrebbe essere una buona idea sperimentare prima con i metodi JavaScript che operano sulle stringhe (charAt, indexOf, slice, substr, substring), ed utilizzare le espressioni regolari se non si riesce a trovare una soluzione alternativa.