Formazione specialistica per sviluppatore front-end

Hello!

Alessandro Pasqualini

Full-stack developer, appassionato di programmazione e sviluppo software in generale, amo le serie tv e i gatti

1

Approfondimento JS

L'operatore ternario

L'operatore ternario (chiamato anche istruzione condizionale) è una "scorciatoia" dell'if in alcuni casi:

```
var variabile;
if (condizione) {
   variabile = 'Condizione vera';
} else {
   variabile = 'Condizione falsa';
}
```

```
var variabile = condizione ? 'Condizione vera' : 'condizione falsa';
```

Creare elementi con jQuery

Capita spesso di dover generare elementi HTML dinamicamente usando Javascript in risposta a certi eventi (callback, richieste/risposte con il server mediante Ajax, etc)

jQuery ci permette di generare elementi utilizzando un sinstassi molto semplice:

```
var elemento = $('<div></div>');
```

Creare elementi con jQuery

```
var elemento = $('<div></div>');
```

jQuery ci ritorna un oggetto che rappresenta il nostro nuovo elemento.

A questo punto possiamo modificarlo ad esempio aggiungendo classi, css, event handler (es. click), etc

Successivamente dobbiamo ricordarci di includerlo all'interno della pagina.

Creare elementi con jQuery

```
var elemento = $('<div></div>');
$('body').append(elemento);
```

Quando jQuery crea un elemento **non lo aggiunge** automaticamente al DOM (e quindi al documento) e di conseguenza l'elemento non viene reso (rendered) dal browser.

Per questo jQuery ci mette a disposizione un paio di metodi (funzioni) per aggiungere l'elemento al DOM.

Creare elementi con jQuery

```
var elemento = $('<div></div>');
$('body').append(elemento);
```

Innanzitutto dobbiamo scegliere un elemento (ad esempio il body) come "punto di ingresso" per inserire il nuovo elemento. E poi chiamare una delle seguenti funzioni:

- **append** aggiunge l'elemento come **ultimo figlio** del "punto di ingresso".
- **prepend** aggiunge l'elemento come **primo figlio** del "punto di ingresso".

Creare elementi con jQuery

```
var elemento = $('<div></div>');
$('body').insertBefore(elemento);
```

Oppure possiamo scegliere se inserire il nuovo elemento prima o dopo il nostro elemento "punto di ingresso", ovvero decidiamo di inserirlo come **sibling** (fratello).

- **insertBefore** aggiunge l'elemento sopra il "punto di ingresso".
- **insertAfter** aggiunge l'elemento sotto il "punto di ingresso".

Templating Javascript

Abbiamo visto che con jQuery possiamo creare nuovi elementi e aggiungerli al DOM con delle semplici chiamate.

Se però dobbiamo creare una struttura HTML "complessa" ciò risulta essere complesso con jQuery dovendo scrivere molto codice.

Javascript ha così introdotto il concetto di templating.

Templating Javascript

Le stringhe templating sono racchiuse dal carattere backtick (` `, che non è l'apice singolo) e possono comprendere più linee.

```
var tpl = `<div class="myclass">
              <p>Sono un template</p>
           </div>`;
```

Naturalmente le stringhe di templating sono solamente stringhe e non un elemento. Dovremo comunque fare ricorso a jQuery per costruire gli elementi ed inserirli nel DOM.

Templating Javascript

```
var tpl = `<div class="myclass">
              <p>Sono un template</p>
           </div>`;

var elemento = $(tpl);
$('body').append(elemento);
```

```
var div = $('<div></div>')
   .addClass('myclass');

var p = $('<p></p>')
   .text('Sono un template');

var div.append(p);
$('body').append(div);
```

Templating Javascript

La vera potenza delle stringhe di templating di Javascript è la possibilità di usare dei **segnaposto**.

Un segnaposto non è altro che un espressione Javascript (del codice javascript).

```
var text = 'Sono un template';
var tpl = `<div class="myclass">
              <p>${text}</p>
           </div>`;

var elemento = $(tpl);
$('body').append(elemento);
```

Templating Javascript

I segnaposti sono composti in questo modo:

```
${espressione}
```

Possono essere inclusi direttamente nella stringa template:

```
var tpl = `1 + 1 = ${1+1}`;
```

Callback

Una variabile è un "contenitore" usato per contenere qualche informazione:

In Javascript è possibile memorizzare molte informazioni differenti:

- Numeri interi (number)
- Numeri a virgola mobile (number)
- Un carattere (string)
- Una stringa, ovvero un insieme di caratteri (string)
- Un valore booleano, ovvero vero o falso (boolean)
- Un insieme di variabili (array)
- Un oggetto (object)
- Una funzione (function)

```
var variabile;
```

Callback

```
var laMiaFunzione = function () {
   console.log('La mia funzione');
}

laMiaFunzione(); // Stampa 'La mia funzione'
```

Quindi è possibile dichiarare una funzione e salvarla all'interno di una variabile. Successivamente è possibile invocarla (eseguirla) semplicemente utilizzando il nome della variabile al posto del nome della funzione.

Callback

```
var laMiaFunzione = function () {
   console.log('La mia funzione');
}

laMiaFunzione();
```

```
function laMiaFunzione() {
   console.log('La mia funzione');
}

laMiaFunzione();
```

Entrambe le funzioni si comportano in maniera completamente identica e possono essere invocate allo stesso modo.

L'unica differenza tra le due dichiarazioni è che **la seconda esplicita il nome della funzione**, mentre **la prima è implicitamente il nome della variabile**.

Callback

Se Javascript permette di salvare le funzioni all'interno di una variabile cosa ci impedisce di passarle come argomenti di una funzione?

Un **callback** è una funzione passata come argomento di un'altra funzione.

Callback

```
// Dichiaro una funzione che accetta un callback
function funzioneDiTest(callback) {
   callback();
}


// Dichiaro una funzione all'interno di una variabile
var laMiaFunzione = function () {
   console.log('La mia funzione');
}


// Chiamo la prima funzione
funzioneDiTest(laMiaFunzione);
```

Callback

```
// Dichiaro una funzione che accetta un callback
function funzioneDiTest(callback) {
   callback();
}


// Dichiaro una funzione all'interno di una variabile
var laMiaFunzione = function () {
   console.log('La mia funzione');
}


// Chiamo la prima funzione
funzioneDiTest(laMiaFunzione);
```

```
La mia funzione
>
```

Callback

A volte è necessario dichiarare una funzione "temporanea", ovvero una funzione che serve solo una volta e non ci interessa salvarla da qualche parte: non la useremo mai più!

Allora al posto di dichiarare una variabile in una funzione e successivamente passare questa variabile ad una funzione come callback possiamo saltare questo passaggio: passiamo direttamente la funzione.

I parametri di una funzione (funtion (paramtro1, parametro2, ..) {...}) sono variabili!

Callback

```
// Dichiaro una funzione che accetta un callback
function funzioneDiTest(callback) {
   callback();
}


// Chiamo la prima funzione passando direttamente la
// funzione come argomento
funzioneDiTest(function () {
   console.log('La mia funzione');
});
```

Callback

```
// Dichiaro una funzione che accetta un callback
function funzioneDiTest(callback) {
   callback();
}


// Chiamo la prima funzione passando direttamente la
// funzione come argomento
funzioneDiTest(function () {
   console.log('La mia funzione');
});
```

```
La mia funzione
>
```

Parametro vs argomento

Un **parametro** è un valore che una funzione prevede che gli venga passato. Quindi è incluso nella sua dichiarazione:

```
function laMiaFunzione(parametro1, parametro2) {

}
```

Un **argomento** è l'effettivo valore passato alla funzione, ovvero il valore che inseriamo nella sua chiamata:

```
laMiaFunzione(argomento1, argomento2);
```

A cosa servono i callback?

Possiamo passare una funzione a come argomento ad un'altra funzione ma perché farlo?

Il codice Javascript che scriviamo viene eseguito principalmente al verificarsi di qualche evento: l'utente clicca su un bottone, la richiesta ajax è finita, la pagina è pronta etc

Quindi non è possibile conoscere a priori il flusso che il programma Javascript seguirà ma saranno gli eventi (o l'utente) che determineranno l'esatta sequenza di esecuzione.

A cosa servono i callback?

Visto che non possiamo conoscere a priori la sequenza (e quando) il nostro codice verrà eseguito dobbiamo prevedere un meccanismo capace di garantire comunque la corretta esecuzione del codice.

Questo meccanismo è fornito dai callback!

Attraverso i callback possiamo prevedere del codice che verrà eseguito solo se un certo evento si presenta.

A cosa servono i callback?

```
$('#pulsante').click(function () {
   console.log('Sono un callback!');
});
```

Quando definiamo un event handler (il codice da eseguire quanto un evento si verifica) stiamo in realtà definendo un callback.

Quello che stiamo dicendo a Javascript di fare è: quando l'utente clicca sul pulsante allora esegui la funzione che ti passo come argomento, ovvero esegui il callback!

3

Esercizio

Esercizio

- Creare un form di registrazione con bootstrap. Il form deve avere nome, cognome, username, email, password e conferma password.
  - Il form deve essere inviato al file form.php
  - L'attributo name dei vari input deve essere firstname, lastname, username, email, password, confirm-password

- Intercettare con jQuery l'invio del form (o del click sul pulsante di registrazione)

- Validare il form di registrazione in modo che il nome e il cognome siano lunghi almeno 3 caratteri, lo username abbia lunghezza compresa tra 7 e 20 caratteri, le due password coincidano e la mail sia valida.

Per validare la mail:

https://stackoverflow.com/questions/46155/how-to-validate-an-email-address-in-javascript

4

Esercizio

Esercizio

- Continuando l'esercizio precedente, l'utente non può inserire uno username già utilizzato.
  - Quando l'utente inserisce lo username, validate "on the fly" lo username con le regole precedenti.
  - Inviate con ajax lo username inserito dall'utente al file username.php che vi dirà se lo username è già stata utilizzato o meno
  - Se lo username è valido mettere una spunta verde, altrimenti una X rossa
- Validate la password in modo che sia sicura (almeno un carattere maiuscolo, un carattere minuscolo, un numero e un carattere speciale)

5

Esercizio

Load more

- L'idea è creare un pulsante che vi permette di caricare altri post (immagine la pagina categoria di un blog)
- Il file posts.php vi ritorna un array JSON contenente un numero il numero di post da voi richiesto
  - http://localhost/post.php?n=10 vi ritorna 10 post
- Dovere creare un bottone che se premuto aggiunge sopra di lui il i post ritornati (dovere costruire anche gli elementi HTML)

6

Esercizio

Scroll infinito

- Il funzionamento è simile al pulsante Load more dell'esercizio precedente solo che questa volta non c'è bottone: ogni volta che scrollate la pagina vengono caricati nuovi post (l'idea è quella del feed di Facebook)

Thanks!

Any questions?

Per qualsiasi domanda contattatemi:
alessandro.pasqualini.1105@gmail.com