emit feltrinelli
ente morale istruzione tecnica

HTML5 and Next RIA Apps

Esplorazione di Javascript

Esplorazione di Javascript

JavaScript e un linguaggio di client-side scripting che permette di creare pagine Web attive, manipolando i tag di una pagina o creandone di nuovi, modificando gli attributi dei tag, gli stili grafici e gestisce gli eventi di ogni tag, visto appunto come un oggetto.

In HTML5 Javascript assume un ruolo FONDAMENTALE, in quanto tutte le nuove features introdotte sono gestibili e utilizzabili SOLO attraverso Javascript (da qui in avanti chiamato semplicemente “JS”).

JS fu introdotto nel 1995 con Netscape Navigator 2.0 e da quin è stato un susseguirsi di miglioramenti; è platform-indipendent cioè l'interprete JavaScript gira all'interno del browser: questo significa che attualmente i diversi browser interpretano in maniera diversa JS e ci sono tre famiglie di interpreti “Mozilla-like” “Webkit-like” e “IE-Like”.

Questo significa che lo stesso codice Javascript non sempre funziona allo stesso modo su tutti i browser, ma queste differenze di interpretazione possono essere superate attraverso opportune corsiderazioni. Fortunatamente di sono diverse librerie Javascript come jQuery che ci semplificano questo tipo di attività.

Inoltre JS non è correlato in alcun modo al linguaggio Java (che è un mondo a parte) ma ne eredita parte della sintassi.

Esplorazione di Javascript

JS è un linguaggio di programmazione orientato agli oggetti, basato su prototipi:
non esiste la distinzione tra classi e istanze, ma solo oggetti che istanziano prototipi.

Il concetto di prototipo è un oggetto che funge da modello da cui prendere le proprietà iniziali di un nuovo oggetto.

Qualsiasi oggetto, poi, può specificare le sue proprietà quando viene creato o durante l'esecuzione.

Metodi di inclusione nella pagina

Il codice JS può essere inserito nella pagina in tre modalità:
- inline: scritto direttamente nella pagina
- file: recuperato da un file esterno
- remota: recuperato da un file esterno che risiede su un altro server (caso dei CDN)

Ecco degli esempi di inserimento:
Inline:

```
<!-- === Inserimento codice "inline" === -->
<script type="text/javascript">
// === Codice Javascript
</script>
```

Esplorazione di Javascript

Metodi di inclusione nella pagina

file: recuperato da un file esterno

```
<!-- === Inserimento codice da file esterno === -->
<script type="text/javascript" src="file-esterno.js"></script>
```

remota: recuperato da un file esterno che risiede su un server esterno

```
<!-- === Recuperato da un server esterno (CDN) === -->
<script type="text/javascript"
        src="http://code.jquery.com/jquery-1.7.1.min.js"></script>
```

Invito a consultare il link per approfondimento:

http://code.google.com/intl/it-IT/apis/libraries/devguide.html

Il codice JavaScript può essere inserito sia nell'header che nel body di un documento:

- le funzioni sono di norma definite nell'header
- il codice inserito nel corpo del documento viene eseguito quando la pagina viene caricata

Esplorazione di Javascript

Metodi di inclusione nella pagina

JS è case sensitive, cioé è necessario rispettare l'uso di caratteri maiuscoli e minuscoli

E’ possibile inserire commenti all’interno del codice con
- la sequenza di caratteri
// testo
che commenta il testo seguente fino alla fine della riga corrente

- la sequenza
/* testo */

che commenta una porzione arbitraria del codice sorgente.

Le variabili sono dichiarate mediante la keyword "**var**"

var pi = 3.1416, x, y, name = "Pippo";

- I nomi delle variabili devono cominciare con un carattere alfabetico o underscore
- Le variabili non hanno alcun controllo di tipo e possono cambiare tipo durante l'esecuzione

Esplorazione di Javascript

Regole di visibilità:

- le variabili dichiarate all'interno di una funzione hanno visibilità limitata a quella funzione;

```
<!-- === Regole di visibilità === -->
<script type="text/javascript">
var txt = 'sono globale';
function avviso()
{
     var txt = 'sono locale';
     alert(txt);
}
</script>
```

- le variabili dichiarate al di fuori delle funzioni sono globali, quindi visibili da ogni punto del documento

```
<!-- === Regole di visibilità === -->
<script type="text/javascript">
var txt = 'sono globale';
function avviso()
{
     alert(txt);
}
</script>
```

Esplorazione di Javascript

Regole di visibilità:

JavaScript ha tre tipi primitivi di dati: **number**, **boolean** e **string** mentre tutti gli altri elementi sono oggetti.

I valori numerici sono sempre memorizzati in virgola mobile

I valori boolean:
- 0, "0", le stringhe vuote, undefined, null, e NaN sono considerati "false"
- tutto il resto è "true"

Le stringhe possono essere racchiuse fra coppie di caratteri ' oppure " (apice singolo o apice doppio);

Le stringhe possono contenere alcune sequenze di escape quali \n (new line), \" (apice doppio), \\ (backslash), etc

La concatenazione di stringhe avviene con l'operatore "+"

```
var concat = "Stringa 1" + 'stringa 2';
```

Esplorazione di Javascript

Operatori Aritmetici e di confronto:

Gli operatori aritmetici sono

+ - * / % ++ –

Osservare bene: tutti i numeri sono floating point e non interi.

Opereratori di confronto o relazionali

< <= == != >= > === !==

== e != valutano l'uguaglianza di valore anche tra variabili di tipo diverso (con le opportune conversioni); invece === e !== considerano gli operandi diversi se sono di tipo diverso.

```
3 == '3'    // restituisce true
3 === '3'   // restituisce false
```

Operatori Logici:

- and: “&&”
- or: “||”
- not: “!”

Esplorazione di Javascript

Definizione delle funzioni:

Una funzione è una unità logica, un'elaborazione separata di codice, che può accettare dei parametri in ingresso e restituire una risposta (oggetto) in uscita attraverso la parola chiave “return”.

E’ buona norma definire le funzioni all’interno della sezione <head> di una pagina HTML

Sintassi:

```
function nomeFunzione(par1, par2, ... parN)
{
    // Codice di elaborazione
    return valore;
}
```

Mentre per richiamare una funzione bastra scrivere il suo nome seguito dai parametri; se questa ha un valore di ritorno, questo può essere catturato in una variabile, ad esempio:

```
var uscita;
uscita=nomeFunzione();
```

Esplorazione di Javascript

Programmazione ad oggetti:

Un oggetto JavaScript à un costrutto che possiede proprietà (attributi) e metodi (funzioni)

La sintassi per accedere alle proprietà di un oggetto è:
oggetto.nomeProprieta

- Per aggiungere una nuova proprietà ad un oggetto basta assegnarle un valore:

```
<script type="text/javascript">
// === Definisco un oggetto
var oggetto = new Object();
// === Definisco gli attributi
oggetto.Nome = 'Gianfranco';
oggetto.Cognome = 'Castro';

// === Definisco un metodo
oggetto.stampaNome = function()
{
    alert(this.Nome);
}

// === Richiamo un metodo
oggetto.stampaNome();
</script>
```

Esplorazione di Javascript

Programmazione ad oggetti:

Per vedere tutti gli attributi di un oggetto, possiamo usare il seguente codice:

```
<script type="text/javascript">
// === Visualizzo gli attributi
for(var i in oggetto)
{
    alert('Proprieta '+i+' = ' + oggetto[i]);
}
</script>
```

Un oggetto in JavaScript è essenzialmente un array associativo costituito da coppie chiave-valore, in cui le chiavi sono i nomi delle proprietà

- Le seguenti due istruzioni sono equivalenti in JS:

```
obj.nome = "Pippo";
obj["nome"] = "Pippo";
```

- Ciò ha importanti conseguenze sulle modalità d'uso degli oggetti e sulle relazioni tra oggetti e array

Per eliminare una proprietà si usa l'operatore delete:

```
delete obj.nome;
```

Esplorazione di Javascript

Programmazione ad oggetti:

La definizione di un oggetto avviene implementando una funzione che prende il nome di costruttore:

```
<script type="text/javascript">
/**
 * Definizione di una classe
 */
function Persona(n, c) {
     // --- La parola-chiave "this" è obbligatoria!
     this.nome = n;
     this.cognome = c;

     // --- Definisco un metodo della classe
     this.metodo = function()
     {
          alert('sono un metodo interno');
     }
}

// === Creo le istanze delle classi
var obj1 = new Persona("Gianfranco", "Castro");
var obj2 = new Persona("Alessandro", "Preziosi");

// === Utilizzo le classi
alert(obj1.nome);
alert(obj2.nome);
obj1.metodo();
</script>
```

Esplorazione di Javascript

Programmazione ad oggetti:

In alternativa si può usare un'istruzione di inizializzazione di oggetto (object initializer):

```
<script type="text/javascript">
var obj = {
    property_1: value_1,
    property_2: value_2,
    ...,
    property_n: value_n
};
</script>
```

Questa è anche la sintassi base per la definizione di oggetti JSON largamente utilizzati nel Javascript “moderno”: JSON è il formato “ideale” per trasportare dati e informazioni tra server e client, essendo molto flessibile e facilmente “assimilato” da Javascript, ma l'utilizzo di JSON richiede particolari considerazioni per essere usato con profitto.
Tra gli argomenti del corso ci sarà una sezione dedicata a JSON.

Esplorazione di Javascript

Programmazione ad oggetti:

Quindi la classe “Persona” precedente può anche essere espressa in questo modo:

```
<script type="text/javascript">
var Persona = {
      'nome' : '',
      'cognome' : '',

      // === Creo il mio costruttore 'custom'
      init : function(n,c)
      {
            this.nome = n;
            this.cognome = c;
      },

      // === Funzione di test
      test : function()
      {
            alert(this.nome+ ' - ' + this.cognome);
      },

      // === Definisco un metodo
      metodo : function()
      {
            alert('sono un metodo interno');
      }
};
// === Utilizzo la classe
Persona.init('Gianfranco', 'Castro');
Persona.test();
Persona.metodo();
</script>
```

Esplorazione di Javascript

Gli array:

E' un dato strutturato che permette la memorizzazione di una sequenza di valori dello stesso tipo.
Si definisce come un nuovo oggetto Array attraverso la sua sintassi canonica, oppure tramite la sintassi JSON:

```
// === Sintassi "canonica"
var nomi = new Array();
nomi[0] = "Antonio";
nomi[1] = "Giuseppe";
nomi[2] = "Cosimo";

// === Sintassi JSON
var nomi = {"Antonio", "Giuseppe", "Cosimo"};


// === ordina l'array numericamente
myArray.sort( function(a, b) { return a - b; } )

myArray.reverse();       // inverte l'ordine degli elementi
myArray.push( ... );     // aggiunge un elemento alla fine dell'array e ne incrementa la dimensione
myArray.pop();           // preleva l'ultimo elemento dell'array e ne riduce la dimensione
myArray.toString(); // restituisce una stringa contenente i valori dell'array separati da virgole
```