Università degli Studi di Bologna
Facoltà di Ingegneria

Tecnologie Web L-A
A.A. 2009 – 2010

Esercitazione 7
DHTML, DOM, Javascript

Tutor:
Ing. Pasini Samuele *samuele.pasini@unibo.it*

Agenda

- Alcune premesse “pratiche”
  - progetto d'esempio
  - risorse statiche e cache dei browser
- Javascript e le applicazioni Web
  - semplici esempi di utilizzi ricorrenti
  - parco giochi

Progetto di esempio

- All'interno dell'archivio ZIP dell'esercitazione, nel direttorio ***progetti***
  - il file ***07_TemplateJavascript.zip*** contiene lo scheletro di un semplice progetto di esempio
  - creato con Eclipse, contiene già tutti i descrittori necessari a essere riconosciuto e configurato correttamente

- Importare il progetto come visto nelle precedenti esercitazioni, senza esploderne l'archivio su file system (lo farà Eclipse)
  - *File → Import → General → Existing Projects into Workspace → Next → Select archive file*

- Eseguirne il deployment

Prima di iniziare

- I browser “fanno cache” del contenuto statico
  - pagine html, script js, fogli di stile css, immagini gif/png/jpeg/jpg/...
  - infati, normalmente non accade che tali risorse cambino tra due invocazioni successive

- Quando si sviluppa, questo comportamento è invece altamente fastidioso e bisogna cercare di ovviare
  - aggiungere i seguenti meta attributi nell'header delle pagine che si stanno modificando:

    <meta http-equiv="Pragma" content="no-cache"/>

    <meta http-equiv="Expires" content="-1"/>
  - **tenersi sempre pronti a svuotare la cache del browser** quando non si osservano gli effetti o le modifiche sperati (anche se si usano i meta attributi)
  - **impostare (se possibile) il browser affinché NON usi la versione in cache delle risorse già scaricate**

Pagine di benvenuto – *welcome.jsp*

- Struttura a *frame*...
  - giusto per sapere che esistono: tutti i maggiori browser li supportano, **ma il loro uso è deprecato**
  - non così l'uso di ***iframe***, invece, data la necessità di poter eseguire richieste cross-domain: (es: gmaps... ricordate?)

- A casa, potete riprovare a ottenere lo stesso effetto con le azioni (request-time) o le direttive (compile-time) di tipo *include* delle pagine JSP
  - come ad esempio nel menu a *tab* del negozio online delle ultime 2 esercitazioni
  - qui non è stato fatto per mantenere le pagine HTML degli script indipendenti e statiche (dal punto di vista del Web server, ovviamente... v. ultima slide).

Pagine di benvenuto – *whoYouAre.html*

- Visualizzata nel frame con nome *“content”* a seguito del codice di *welcome.jsp*

- Esempio di browser detection
  - occorre spesso diversificare il codice Javascript eseguito in funzione del browser rilevato (anche per i fogli di stile succedeva qualcosa di simile... ricordate?)
  - tipicamente, le operazioni di "scelta" non vengono fatte "a mano" ed esplicitamente in ogni pagina, ma richiamando funzioni di libreria che nascondono tale complessità
    - sviluppate da voi stessi una tantum e poi invocate da parte degli script delle pagine
    - fornite da terze parti (es: *jquery*)

- Disattivate ora l'uso di javascript nelle preferenze del browser e aggiornate la pagina
  - differenze?
  - riattivate Javascript, ora!

Definizione e invocazione di funzioni - *helloworld.html*

- Invocazione di funzioni man mano che il browser processa, renderizza e interpreta il sorgente della pagina
  - le funzioni invocate, `alert()` e `confirm()`, sono predefinite
  - la loro definizione e implementazione è fornita dall'interprete Javascript presente nel browser

- La definizione di una funzione non ne comporta l'immediata esecuzione, ma la rende disponibile nel proseguimento della pagina, ai fini dell'invocazione:
  - dall'interno di uno script *in linea* nella pagina
  - dall'interno di altro codice Javascript
  - associata ad eventi DHTML

- in evidenza:
  - i messaggi popup interrompono il rendering
    - diversi browser, tuttavia, tentano di schedulare il rendering e l'esecuzione di codice javascript come processi paralleli, quando possibile
    - provate (a casa su) chrome o safari per farvi un'idea delle differenze di comportamento che emergono
  - l'uso dell'alternanza tra apice singolo ' e virgolette “ permette di scrivere HTML ben formato anche all'interno del codice Javascript, senza utilizzare escaping
    - apice singolo e virgolette sono entrambi delimitatori di stringa validi in Javascript e **all'interno di una coppia di delimitatori di un tipo si possono usare quelli dell'altro tipo** senza bisogno di escaping

Interazione tra Javascript e il DOM - *javascript-dom.html*

- L'esecuzione di una funzione che interagisce con il contenuto visualizzato deve...

ATTENDERE IL COMPLETAMENTO DEL CARICAMENTO DEL CONTENUTO

...da parte del browser

- Sembra banale, eppure...

Reazione ad eventi DHTML - *javascript-events.html*

- in questa pagina
  - l'evento ***onload*** è sfruttato per agganciare funzioni Javascript a parti del DOM, in corrispondenza degli eventi DHTML da essi supportati
  - un'altra funzione (definita nella pagina stessa) viene associata ad un evento DHTML direttamente nel tag interessato

- in evidenza:
  - all'interno del codice di una funzione Javascript è possibile definire nuove funzioni ed assegnarle

In Javascript, infatti, le funzioni sono oggetti di prima calsse.

Non c'e' alcuna differenza in Javascript tra un numero, una stringa, e una funzione. Tutti questi tipi di dati possono essere passati come argomento, memorizzati in variabili e ritornati come valore di ritorno di una funzione.

Primi problemi (1) - *rollover.html*

- L'effetto di evidenziazione al passaggio del mouse non sempre e' ottenuto modificando il colore di sfondo e/o del font
  - spesso si usano piccole immagini, sostituite tra loro in corrispondenza degli eventi relativi al mouse
  - l'effetto prende il nome di 'roll over'

- In questa pagina, 4 modi diversi di ottenere lo stesso effetto
  - tramite CSS e pseudoclassi
  - tramite DOM e DHTML
  - tramite funzioni Javascript

- ma.... l'ultima funzione non va!
  - a differenza del codice JSP (compilato), quello Javascript è interpretato e la presenza di errori si scopre solo al momento di eseguire il codice stesso!!
  - infatti, non ci siamo accorti del problema finché non abbiamo "fatto girare" (o, almeno, provato a far girare) la funzione incriminata
  - inoltre, il servlet container fallisce la traduzione e compilazione in servlet e ci restituisce un errore (500) e il relativo stackTrace dove guardare: il browser, invece, prosegue “a testa bassa” se qualcosa non va e semplicemente "inibisce" le parti di codice non corrette

Primi problemi (2) - *rollover.html*

- Sì, ma... e adesso? un indizio sul problema?
  - ore e ore a spulciare il codice
  - uso di milioni di `alert()` per mostrare popup in cui “loggare” lo stato di variabili e oggetti per capire dove nasce il problema
  - usare un tool di sviluppo piu' avanzato!

- E' il momento di Firebug:
  - “logging” dei messaggi di errore Javascript su una specie di console di stderr
  - esecuzione del codice Javascript in modalità debug
  - ...e molto di più

- F12 o “click” sullo scarafaggio in basso a destra per attivare Firebug
  - interagire con la pagina per lanciare l'esecuzione della funzione sbagliata e leggere i messaggi di errore nella console
  - cosa correggere??
  - una volta che la funzione esegue... collocare breakpoint al suo interno e seguirne l'esecuzione in modalità debug!

- Non stavate usando IE, vero?

Validazione di form lato client - *validate.html*

- Tipico utilizzo di Javascript per evitare di...
  - formulare HTTP request (magari anche pesanti) destinate a fallire
  - evitare di consumare banda
  - scrivere pagine piu' “responsive” e migliorare l'esperienza utente

- Accesso agli elementi di una form, via DOM, e verifica di correttezza: seguiamo insieme l'esecuzione in modalità debug
  - apertura di firebug, tab script, selezione del sorgente Javascript che interessa
  - collocazione dei breakpoint
  - interazione con la pagina per scatenare l'invocazione del codice Javascript (click sul pulsante, nel nostro caso)

- In evidenza :
  - "esternalizzazione" delle funzioni Javascript in appositi file ***.js***, richiamabili da piu' pagine
  - è possibile (e anzi.. è prassi comune) utilizzare i valori di ritorno booleani delle funzioni Javascript per bloccare o permettere l'effettiva submission di una form

Javascript e l'oggetto RegExp (1)

- In Javascript le espressioni regolari sono incarnate dall'oggetto RegExp
- Due modi di dichiarare tale oggetto
  - tramite la scrittura del suo *literal, nella forma*

    ```
    var re = /pattern/modificatori
    ```

    - la regex vera e propria è compresa tra i caratteri **/**
    - I modificatori sono rappresentati da zero o più delle lettere **m i g**
    - utilizzando questa forma, la regex è cablata nel codice all'atto del caricamento dello script
  - tramile il costruttore dell'oggetto RegExp

    ```
    var re = new RegExp("pattern","modificatori")
    ```

    - I modificatori sono sempre rappresentati da zero o più delle lettere **m i g**
    - utilizzando questa forma, pattern e modificatori possono essere il risultato di una valutazione run-time, ad esempio:

```
var re = new RegExp(window.prompt("Please input a regex.","yes|yeah"),"g");
```

- I modificatori comandano il seguente comportamento
  - g = la regex si applica a tutta la stringa candidato
  - i = il confronto è case-insensitive
  - m = il confronto è multilinea (provare....)

Javascript e l'oggetto RegExp (2)

- Sono quindi possibili diversi modi di applicare un'espressione regolare a una stringa candidato; ad esempio...

  - **`RegExp.exec(string)`** restituisce i possibili match
    var match = /s(amp)le/i.exec("Sample text") → il risultato, in *match*, è la coppia di stringhe *["Sample","amp"]*

  - **`RegExp.test(string)`** restituisce l'esito booleano del confronto con la stringa candidata
    var match = /sample/.test("Sample text") → il risultato, memorizzato in *match*, è *false*

  - **`String.match(pattern)`** restituisce le sottostringhe che soddisfano il pattern
    var str = "Watch out for the rock!".match(/r?or?/g) → restituisce tutte le occorenze, data l'indicazione di 'g'

  - **`String.search(pattern)`** restituisce l'indice della prima sottostringhe che soddisfa il pattern
    var ndx = "Watch out for the rock!".search(/for/) → il risultato, memorizzato in *ndx*, è pari a *10*

  - **`String.replace(pattern,string)`** sostituisce le occorrenze del pattern con la stringa passata in arg
    var str = "Liorean said: My name is Liorean!".replace(/Liorean/g,'Big Fat Pig') → il risultato, memorizzato in *str*, è *"Big Fat Pig said: My name is Big Fat Pig!"*

  - **`String.split(pattern)`** spezza la stringa in un array, usando il pattern come separatore
    var str = "I am confused".split(/\s/g) → il risultato, memorizzato in *str*, è *["I","am","confused"]*

- Provate a modificare *validate.html* e *validator.js* per riprodurne i risultati tilizzando le espressioni regolari!

Modifica della visibilità del contenuto - *hideAndSeek.html*

- Un tipico effetto grafico per ottenere contenuto dinamico all'interno di una pagina
  - tutto il contenuto informativo è scaricato all'atto della richiesta, ma solo una parte è immediatamente visibile
  - attraverso gli eventi DHTML scatenati dalle azioni dell'utente si aggancia l'esecuzione di codice Javascript che modifica gli attributi di stile degli elementi interessati
    - *display: block;* → l'elemento è mostrato E occupa spazio nella pagina
    - *display: none;* → l'elemento NON è mostrato E NON occupa spazio
  - intervenendo sull'attributo
    - *visibility: hidden;*

    si sarebbe invece ottenuto l'effetto di nascondere l'elemento, ma lasciare “vuoto” lo spazio che esso avrebbe occupato.

Parco giochi (1) - *calculator.html*

- Simile all'esempio presente nelle slide di teoria, serve qui a imprimere il concetto che **IL BROWSER E L'INTERPRETE JAVASCRIPT CHE NE LEGGE IL DOM RAGIONANO ENTRAMBI A STRINGHE**
  - non sta scritto da nessuna parte che un campo di input testuale sia destinato all'immisione di numeri
  - se non viene forzato a “valutare” l'espressione rappresentata dall'input testuale, l'interprete Javascript ne considera il valore come stringa

- Come sistemare?
  - Sbirciate nel codice...
  - Ovviamente non aprendo i file, ma con Firebug!!!

Parco giochi (2) e nuovi problemi - *relocator.html*

- Un simpatico giochino...
  - che però richiede di diversificare il proprio codice in funzione del browser su cui viene eseguito :(
  - l'esempio © 2006 era pensato per riconoscere IE e Mozilla, nelle versioni di allora...

- I browser introducono spesso estensioni proprietarie...
  - attributi di stile (es: *moz-.....*)
  - funzioni Javascript predefinite
  - struttura del DOM

  ...e, viceversa, mancano spesso di fornire pieno supporto alle specifiche del W3C
  - Confrontate i risultati del test ACID3 che potete ottenere con i diversi browser: Firefox, Chrome, Safari, IE, Opera, ...
    - *http://acid3.acidtests.org/*

- E' quindi molto facile scrivere pagine che sfruttano tali estensioni e/o parti delle specifiche non ancora supportate da tutti
  - W3C e altri siti forniscono utili riferimenti in merito a cosa è supportato da chi
  - link nella pagina...

Parco giochi (3) - *breadcrumbs.html*

- Accesso programmatico all'oggetto *window* e alle proprietà della pagina corrente
  - la location corrente viene elaborata per produrre un insieme di link diretti alla pagina principale di ciascun direttorio attraversato per raggiungerla, in stile “mollica di pane”
  - da leggere ed assorbire con calma
  - ottimo per esercitarsi a casa, magari con Firebug,

Una piccola nota conclusiva

- Abbiamo lavorato sul progetto d'esempio utilizzando
  - il browser
  - il Web server
  - il protocollo HTTP

- In realtà...
  - ...alcune pagine del progetto (i menu, la pagina con i frameset, ...) sono basate su tecnologia JSP e richiedono il Servlet Container per funzionare
  - ...ma le pagine oggetto degli esempi Javascript (cioè quelle all'interno del direttorio ***pages***) richiedono soltanto la presenza dell'interprete Javascript per poter essere visualizzate
    - sono infatti risorse statiche con estensione **.html**
    - richiamano altre risorse statiche (immagini **.gif/.png**, script **.js**, fogli di stile **.css**) attraverso path relativi da cui il browser ottiene URL completi basandosi sull'URL della pagina visualizzata
    - potete quindi aprire ciascuna di esse direttamente nel browser, utilizzandone l'URL su file system (**file://...**), senza lanciare il Web Server e accedere via protocollo http (**http://...**)