Come funziona un browser

I browser, questi sconosciuti...

Roberto Castaldo
Rcastaldo@webaccessibile.org

Il browser, questo sconosciuto...

- E' un applicativo piuttosto complesso
  - Flessibilità nell'interpretazione e visualizzazione di documenti e pagine
- Principi di funzionamento relativamente semplici

Programmazione tradizionale

- La programmazione tradizionale è composta da una serie di **procedure** (blocchi di codice) che tramite istruzioni condizionali o ricorsive trattano una serie di dati con degli specifici tipi

Programmazione ad oggetti

- I tipi nei linguaggi orientati ad oggetti sono realmente degli **oggetti** con delle specifiche ben precise
  - Si tratta di oggetti astratti quasi completamente indipendenti dal sistema

Gli oggetti

1. Si scrive una **Classe** = il modello dell'oggetto
2. Si inseriscono una serie di **proprietà** che l'oggetto deve possedere e che non sono in genere visibili al di fuori dell'oggetto
3. Si inserisce un **costruttore** che è una funzione che viene chiamata alla creazione di un nuovo oggetto
4. Si inseriscono i vari **metodi** che sono una serie di funzioni che servono a trattare le proprietà interne dell'oggetto

Una classe…

```
class cPersona extends object
{
 private var nome:cNome;
 private var cognome:cCognome;

//classe costruttore

 public function cPersona(var tNome:cNome,
         var tCognome:cCognome)
       {
       this.nome = tNome;
       this.cognome = tCognome;
       }

public function datiAnagrafici()
{
 return (this.nome.toString()+
 " "+this.cognome.toString())
}
```

Programmazione ad oggetti

- Questo tipo di programmazione risulta molto efficiente
- Permette al sistema di mantenere separati ed indipendenti tutti i suoi moduli che si occupano di gestire le operazioni a loro assegnate

Ereditarietà

- E' la possibilità di estendere una classe sulla base di un altra classe ereditandone tutte le proprietà ed i metodi e permettendo così un ulteriore flessibilità
  - *Per esempio, la classe "orologio analogico" eredita le proprietà della classe "orologio"*

HTML e linguaggi ad oggetti

- L'HTML non è un vero linguaggio di programmazione ma solo di marcatura
- Però le sue strutture dati, derivate dall'SGML, sono molto simili a quelle del paradigma ad oggetti

HTML e linguaggi ad oggetti

Ciascun nodo di HTML descrive un reale oggetto di programmazione con:

- Un **tipo** = il nome del tag
- Le **proprietà** dell'oggetto = le proprietà dell'html
- I **metodi** = il funzionamento dello stesso

HTML e linguaggi ad oggetti

- Dall'introduzione di *javascript* questa modellizzazione è stata introdotta anche all'interno dei browser
- Il **W3C** ha standardizzato il modello con il **DOM** *(Modello ad Oggetti per Documenti)* per i linguaggi modellati ad oggetti (sulla base di XML)

HTML e linguaggi ad oggetti

- In pratica all'interno del browser si è creato un oggetto che modella un nodo generico HTML, chiamato "**HTMLobject**"
  - Da questo sono poi stati derivati un'infinità di altri oggetti che descrivono tutte le marcature di HTML

HTML e linguaggi ad oggetti

- **HTML** = insieme di oggetti tipizzati e annidati tra loro che formano il cosiddetto albero DOM
  - Il modello DOM comprende anche una serie di metodi che fungono da interfaccia di trattamento e che ci permettono di trattare dati e strutture

Le interfacce del browser:

In realtà le interfacce usate dal browser per trattare l'HTML sono due:

- Il DOM
- Il SAX (Simple API for XML)
  - *Entrambe le interfacce sono nate per il trattamento di XML*

Il DOM (Document Object Model)

- Crea un albero di tutti i nodi in un area di memoria
- Ogni oggetto mette a disposizione una serie di metodi per la ricerca e la navigazione tra i vari nodi e le proprietà

Il DOM (Document Object Model)

- Mette inoltre a disposizione dei metodi che permettono l'**inserimento**, la **modifica**, la **copia** e l'**eliminazione** di qualunque metodo o proprietà

Il DOM (Document Object Model)

- Il punto debole del DOM è che più è grande il numero di nodi più è grande la memoria occupata e quindi più è lento il sistema
- *Questo principio vale anche per i browser dato che usano DOM per il trattamento dei dati HTML*

SAX (Simple API for XML)

- E' molto più elegante
- Usa la memoria al contrario

SAX (Simple API for XML)

- Si memorizzano in memoria delle sequenze pattern associate a delle istruzioni, e si fa scorrere l'albero DOM nodo per nodo senza doverlo caricare in memoria

SAX (Simple API for XML)

- Se il pattern dei nodi corrisponde, le istruzioni vengono eseguite sul nodo selezionato e poi si va avanti

SAX (Simple API for XML)

- La memoria non si sovraccarica
- Però il SAX non può essere usato in real time
  - *Per ogni operazione bisogna farsi ripassare l'intero documento*
  - *Infatti SAX è stato pensato per i linguaggi di trasformazione come XSL o CSS*

Ma come funziona un browser?

- Come una catena di montaggio
- E' composto da tante piccole parti indipendenti che in sequenza trattano i dati
  - *Questa sequenza è quasi identica in tutti i browser e si svolge tramite dei passaggi quasi standard:*

1 - Caricamento dati

- Il browser carica un documento HTML come fosse normale testo in un oggetto in memoria.

1 - Caricamento dati

- In questa fase il browser non sa distinguere il markup dal contenuto e quindi considera il testo una lunga stringa
- Man mano che il contenuto viene caricato viene in tempo reale passato ad un altro oggetto in memoria che procede alla fase di **parsing** della stringa

2 - Parsing e correzione errori

- Il browser si legge tutte le istruzioni contenute nel testo le corregge e le trasforma in oggetti DOM specifici con impostazioni standard

2 - Parsing e correzione errori

- L'oggetto di parsing opera sulla stringa, che viene:
  - Trasformata in una collezione di caratteri (array)
  - Al suo interno vengono ricercati i singoli tag
  - Quindi la collezione viene splittata e annidata in modo da avere una serie di collezioni annidate *(detti array multidimensionali)*

2 - Parsing e correzione errori

- Ora il browser procede ad un secondo parsing più raffinato:
  - Identifica il linguaggio dichiarato nella dtd in modo da conoscere il linguaggio che dovrà trattare
    - Se non è dichiarata la dtd cerca di intuirlo
  - In base al linguaggio carica una libreria di riferimento e mappa il markup trasformando le collezioni annidate in oggetti che rappresentano gli oggetti html formando così **l'albero DOM in memoria**.

2 - Parsing e correzione errori

- Durante questa fase il browser effettua anche una scansione del codice cercando eventuali **errori**
- Se ne trova, effettua un operazione di interpretazione cercando di capire cosa voleva fare il programmatore (***interpretation***)

2 - Parsing e correzione errori

- Nel nuovo albero DOM sono stati inseriti tutti i contenuti, ma anche rimappati gli attributi definiti a livello di html, anche quelli non settati
  - In realtà tutti gli attributi vengono settati e tutti gli elementi possiedono dei valori di default sia per le proprietà, sia per gli eventi sia per gli stili
- A questo punto vi è la **validation** che testa il neoformato albero DOM
  - Il browser effettua un ultima verifica sull'integrità dell'albero controllando se l'insieme dei nodi risulta ben formato (well formed), cioè che i tag siano annidati in maniera corretta

3 - Rendering

- Il browser renderizza e visualizza la struttura dati con gli attributi di default dati dall'albero DOM
  - *Il **rendering** continuerà perennemente da qui in avanti*

3 - Rendering

- Questo **rendering** lavora in due modalità:
  - **Quirks mode**
    - Dovrebbe rappresentare strutture non propriamente valide
  - **Standard mode**
    - Dovrebbe rappresentare ottimamente le strutture valide.
- Lui scansiona nodo per nodo gli oggetti DOM che rappresentano le singole marcature e li rappresenta sullo schermo come normale testo

3 - Rendering

- In seguito analizza gli **attributi di stile** e li applica sullo schermo
- In questo modo, oggetto per oggetto, HTML si va a formare tutta la rappresentazione grafica del sito impostata a livello di HTML

3 - Rendering

- Il renderer ha anche la peculiarità di operare in maniera **autonoma** e **ricorsiva**
  - In pratica ripassa continuamente tutta la struttura in cerca di modifiche da rappresentare

3 - Rendering

- A questo punto la nostra struttura di programmazione e di interfaccia DOM è correttamente formata e rappresentata con tutti gli attributi settati (o di default)
  - Da qui in avanti il browser tratterà solo una serie di trasformazioni “funzionali”

4 - Trasformazione

- Il browser interpreta ed esegue i documenti di trasformazione e modifica l'albero DOM

4 - Trasformazione

- Il browser è programmato per **caricare altri file in contemporanea** quando incontra dei tag specifici che li richiamano
  - Stiamo parlando di tutti quei documenti come i file **javascript**, **css**, **xsl**, immagini, ecc… che vengono caricati contemporaneamente con l'HTML
  - *La fase di trasformazione prende in carico i file del tipo CSS ed XSL*

4 - Trasformazione

- Sono file scritti con una sintassi conosciuta dal browser che gli permette di modificare la struttura DOM dell'HTML in memoria
  - In pratica quando un tag di richiamo CSS viene trasformata nel corrispettivo oggetto DOM, viene richiamato un secondo oggetto che carica il documento CSS ed attiva una procedura di trasformazione

4 - Trasformazione

- Questa procedura è composta da una serie di funzioni che:
  - Memorizzano tutti i pattern (selettori) presenti nel CSS
  - Si passano in maniera ricorsiva l'intera struttura DOM presente alla ricerca di sequenze DOM che corrispondano al pattern
- Se un pattern trova coincidenza nella struttura gli attributi definiti al suo interno vengono copiati tramite i metodi del DOM all'interno dell'oggetto DOM selezionato

5 - Dinamismo

- Il browser carica in memoria le strutture dati supplementari date dai **linguaggi dinamici** e di **scripting** eseguendone le istruzioni
  - Questa fase si attiva solo in presenza di script

5 - Dinamismo

- A differenza dei linguaggi di trasformazione, **i linguaggi di scripting sono veri è propri linguaggi di programmazione**
  - Permettono la creazione di proprie procedure, tipi ed oggetti permettendo al programmatore di realizzare veri e propri software online
- Inoltre i linguaggi di scripting sono **persistenti** e non ricorsivi
  - Le operazioni non vengono eseguite se non richiamate

5 - Dinamismo

- Javascript implementa nei vari browser una serie di **metodi** che permettono di accedere alla struttura DOM della pagina aggiungendo **azioni** basate su delle condizioni fisiche sulla pagina chiamate **eventi**
  - Gli eventi sono delle funzioni di default, previste negli oggetti DOM che rappresentano l'HTML, che si attivano al verificarsi di determinate azioni (fisiche o virtuali) come il click del mouse sull'oggetto rappresentato visivamente dal renderer o il caricamento dell'oggetto stesso

5 - Dinamismo

- I **metodi** e gli **eventi** DOM per interfacciarsi con la struttura DOM dell'HTML sono moltissimi e sono definiti in 3 livelli DOM definiti dal W3C

6 - Uso

- Il browser visualizza le strutture secondo le regole fissate nell'albero DOM e in caso di dinamismo esegue le operazioni associate agli eventi

6 – Uso: dominanza

- Abbiamo visto che l'albero DOM viene creato, settato e modificato da diversi oggetti con metodologie e tempistiche differenti
- In genere (anche se in alcuni browser cambia) la precedenza, **cioè chi ha diritto a dettare l'ultima modifica**, la detiene il linguaggio di scripting adottato
  - Anche qui non vi sono certezze.
  - In IE, se si adotta Javascript e VBScript nella stessa pagina, la dominanza c'è l'ha VBScript mentre in altri browser con tecnologie di scripting proprietarie la cosa cambia ancora, quindi non resta che provare

6 – Uso: dominanza

La sequenza di dominanza più o meno universale è:

1. Linguaggi di scripting
2. Fogli di stile a cascata CSS
3. Fogli di stile estensibili XSL/XSL-FO
4. Marcature HTML proprietarie
5. Attributi default degli oggetti DOM

6 - Uso

- Ma non è ancora finita…
- Il browser istanzia una serie di proprietà e strumenti **proprietari**
  - Tra questi ricordiamo in IE la copiatura dell'albero DOM renderizzato nell'oggetto di sistema **MSAA** che poi verrà usato dagli screen reader

7 - Distruzione

- Al termine della consultazione del documento la memoria viene svuotata ed è pronta ad ospitare nuove strutture dati

Il Doctype Switching

Come i browser applicano (o non applicano) gli standard

Commissione interministeriale permanente
per l'impiego delle ICT a favore delle categorie deboli o svantaggiate
Segreteria tecnico scientifica

Roberto Castaldo
r.castaldo@iol.it

Browser, Tag Soup e Standard

I browser dovrebbero tutti sapere interpretare gli standard del W3C

- Ma come comportarsi in presenza di Tag Soup?
  - A tutt'oggi la maggior parte dei siti è prodotta con tag proprietari e tips non standard

Browser, Tag Soup e Standard

- Todd Fahrner *(cofondatore del Web Standards Project e membro dei gruppi di lavoro CSS e HTML del W3C)* suggerì di inserire uno switch che "comunicasse" ai browser se il codice era da interpretare secondo gli standard

Browser, Tag Soup e Standard

- La pagina inizia con un doctype **corretto**:
  - L'autore basa la sua pagina sugli standard
  - Il browser cerca di interpretare quella pagina secondo gli standard web
- La pagina era **priva** di doctype (o doctype **errato**)
  - Il browser dovrà usare la modalità "non conforme agli standard“ cercando di “imitare” il comportamento dei vecchi browser

Browser, Tag Soup e Standard

- Il problema è che, anche sul doctype switching, **non tutti i browser si comportano allo stesso modo**
  - Ciascuno di essi si comporta in maniera diversa in presenza dello stesso doctype

Internet Explorer

Ms IE ha soltanto due stati:

- Standard
- Quirks
  - E' quella non conforme agli standard
  - IE 5 possiede solo la modalità quirks
  - In presenza di un prologo xml, IE6 visualizza la pagina in modalità quirks

Browser Gecko (Mozilla)

I browser Gecko (Mozilla ad esempio) hanno tre modalità:

- Quirks
  - NN4 ha solo la modalità quirks
- Quasi-Standards
  - (equivale alla Standards su IE)
- Standards
  - (leggermente differente dalla prima)

Strict e Quasi Strict???

La differenza principale è:

- **Standard mode:**
  - *Img* è un elemento di tipo inline
- **Quasi Standard mode:**
  - *Img* è un elemento di blocco

Browser, Tag Soup e Standard

- Inoltre, pur essendoci un doctype corretto il codice potrebbe ugualmente essere non conforme!
  - L'interruttore è la dichiarazione di doctype, NON il codice validato secondo gli standard

Per esserne certi...

Per controllare lo stato in cui sta funzionando il vostro browser:

- **Mozilla**
  - *Strumenti* > *Informazioni sulla pagina* attiva la information page
- **IE**
  - `javascript:alert(document.compatMode);` genera un alert box
    - *BackCompat* o *QuirksMode* indicano il funzionamento in modalità quirks
    - *CSS1Compat* segnala la modalità quasi standard

Quirks mode...

- Funzionalità comuni a tutti i browser:
  - In presenza di valori numerici inseriti senza unità di misura, questi vengono interpretati come espressi in pixel
  - In presenza di valori dei colori in esadecimale ma senza il carattere #, questi vengono comunque accettati

Sitografia

- www.w3.org
- www.webstandards.org
- www.alistapart.com
- www.webaccessibile.org/argomenti/argomento.asp?cat=490
  - *La trilogia del Browser, di Luca Mascaro*