AJAX

Dario Bottazzi
Tel. 051 2093541,
E-Mail: dario.bottazzi@unibo.it,
SkypeID: dariobottazzi

Recap.

- Ci sono domande?
- Esame
- Proponete esercizi sui temi del corso e condividiamo le soluzioni

Rich User Experience

- Quando lavoriamo con applicazioni desktop siamo abituati ad un elevato livello di interattività con l'applicazione
- Le applicazioni reagiscono in modo rapido ed intuitivo ai comandi
- Non è tipicamente necessario premere un pulsante o visitare un link perché vengano generati e gestiti eventi della GUI

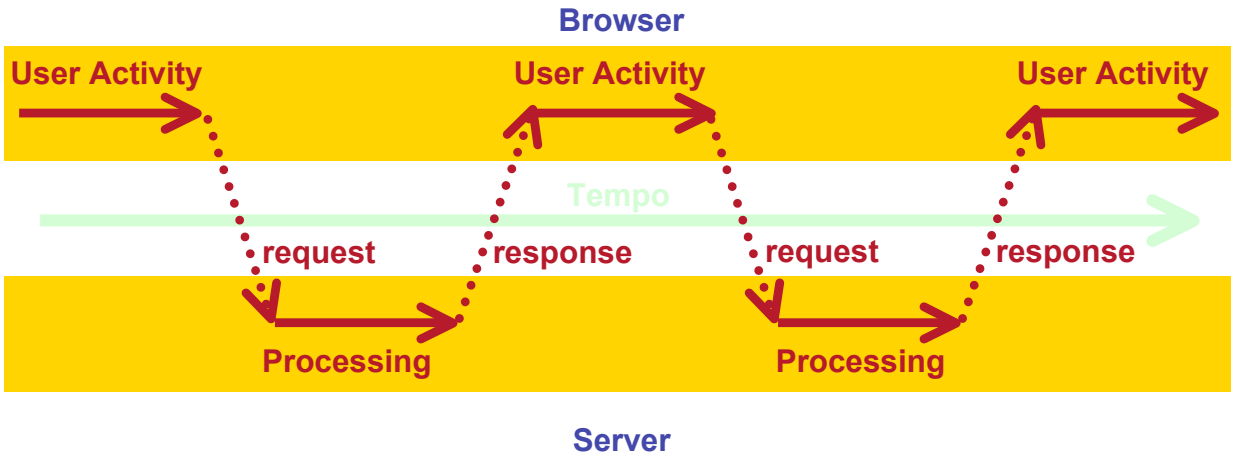
Applicazioni Web Tradizionali

- Dal punto di vista dell'utente le pagine web tradizionali, sia statiche, sia dinamiche suggeriscono un modello di interazione rigido
  - Modello "Click, wait and refresh"
  - È necessario il refresh della pagina da parte del server per la gestione di qualunque evento, es. submission di dati tramite form, visita di un link per ottenere informazioni di interesse, …
- Modello sincrono di interazione fra utente ed applicazione Web
  - L'utente effettua una richiesta ed attende la risposta da parte del server
- La logica di navigazione delle pagine è rigida e determinata dal server

Problemi delle Applicazioni Web Tradizionali

- Interrompono l'interazione fra l'utente ed il documento Web
  - Gli utenti non possono fare alcuna azione mentre sono in attesa di un documento richiesto al server
- Ad ogni refresh o nuova richiesta le pagine vengono caricate completamente
  - Questo carica inultilmente il server
  - È fastidioso per l'utente: necessario rileggere la pagina, perdita della posizione
- Mancanza di feedback sulle attività dell'utente. È necessario attendere la risposta del server

Modello di Interazione Tradizionale



L'interazione con l'utente è interrotta mentre stiamo caricando la pagina

Documenti Attivi

- Sono state proposte diverse tecnologie
  - Java Applet
  - Macromedia Flash
  - JavaScript
  - AJAX

Applet

- Semplici applicazioni Java che vengono incapsulati nei documenti HTML
- Possono utilizzare tutte le API di Java
  - Gestione degli stream di dati, interfaccia grafica, multi-threading
- Schema chiaro e stabilito con precisione
- Tempo per il download spesso significativo
- Difficile riusare la stessa applet adattandola a diversi layout della pagina

Macromedia Flash

- Originariamente progettato per mostrare sequenze animate interattive
- Programmato in ActionScript (linguaggio proprietario simile a JavaScript)
- Esempi di implementazione di Flash sono
  - Macromedia Flex
  - Laszlo (open source)
- Eccellente gestione della grafica vettoriale
- Il browser ha bisogno di un plugin
  - Spesso è necessario aggiornare il plugin

JavaScript

- In realtà DHTML = JavaScript + DOM + CSS consente di creare pagine interattive
- Il difetto principale è ancora l'accoppiamento sincrono fra client e server
  - Per aggiornare i contenuti è necessario ricaricare la pagina
  - Consente però l'implementazione di semplici script per
    - Aggiornare in contenuto delle pagine "on-the-fly" utilizzando DOM
    - Implementare script per la validazione di semplici form

AJAX

- Supera le limitazioni di JavaScript
- AJAX non è un acronimo ma alcuni, con qualche ragione, definiscono AJAX come

Asynchronous JAvascript and Xml

- AJAX è basato su tecnologie standard
  - JavaScript + DOM + XMLHttpRequest
  - XML
  - HTML
  - CSS
- Numerosi toolkit disponibili (oltre 150 i principali)

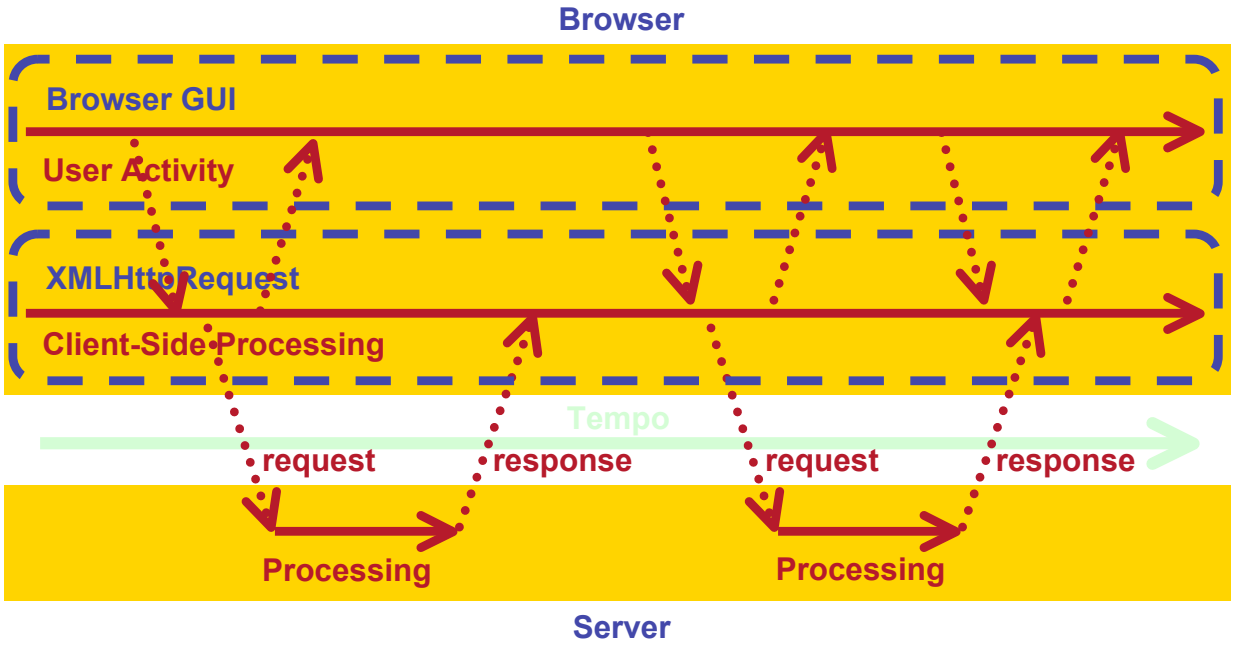
Obiettivo di AJAX

- AJAX punta a supportare applicazioni user friendly
- L'idea alla base di AJAX è quella di consentire agli script JavaScript di interagire direttamente con il server
- Si usa l'oggetto JavaScript XMLHttpRequest
  - Consente di ottenere dati dal server senza la necessità di ricaricare l'intera pagina
  - Comunicazione Asincrona fra client e server: il client non interrompe l'interazione con l'utente anche quando è in attesa di risposte dal server

XMLHttpRequest

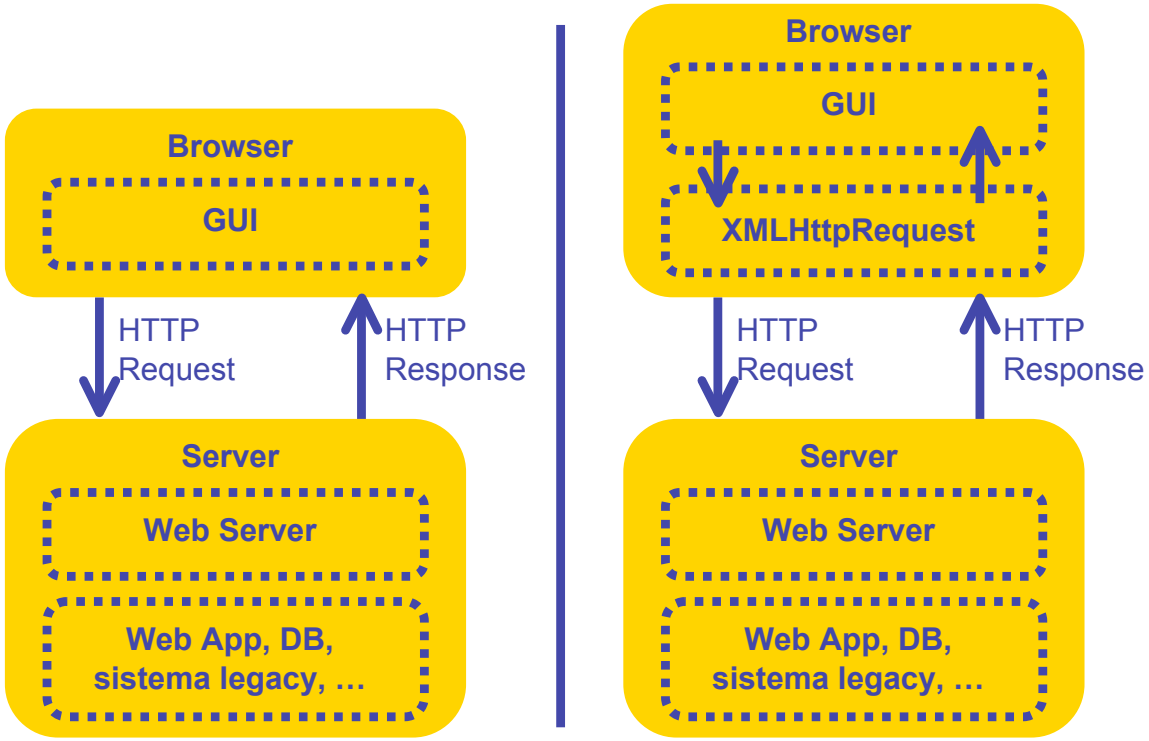
- Oggetto JavaScript introdotto da Microsoft in Internet Explorer (che però non ha sfruttato il vantaggio sui competitors)
- XMLHttpRequest è ormai disponibile in tutti i browser moderni
- Utilizzo di XMLHttpRequest reso popolare da Google (Google Suggest) nel 2005
  - Mentre si scrive la chiave di ricerca nella form di Google il motore restituisce una lista di suggerimenti

Modello di Interazione AJAX



L'interazione fra utente e pagina non viene interrotta ed il caricamento di contenuti avviene in backgroud

Architetture Tradizionali Vs AJAX



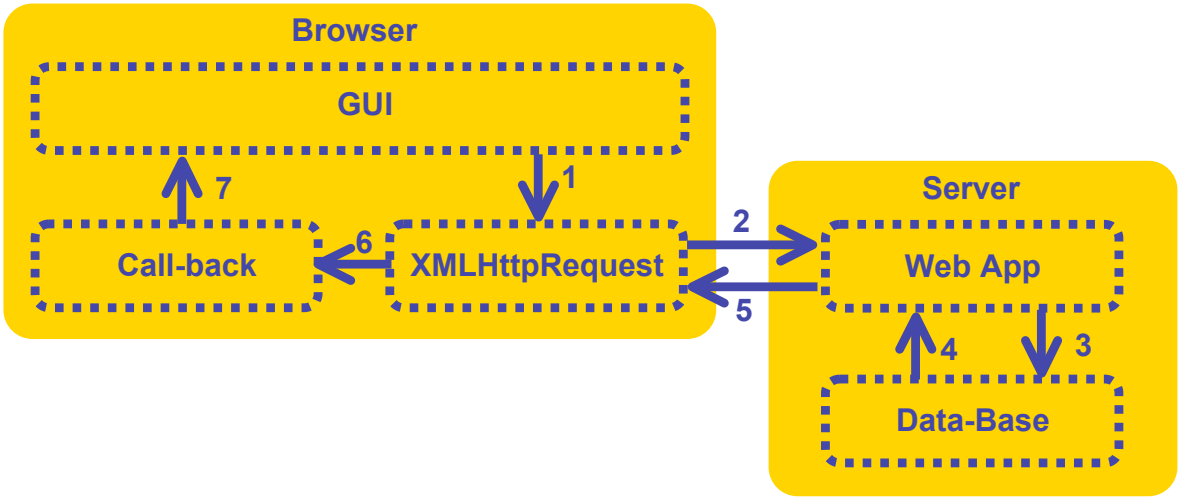
Elaborazione delle Richieste AJAX da Parte del Server

- Il server non deve mettere in atto particolari accorgimenti per consentire ai documenti Web l'utilizzo di AJAX
  - Riceve HTTP Request di documenti (GET o POST) e risponde con HTTP Response
  - Utilizzo di supporti server-side convenzionali Servlet, PHP, Ruby on Rails, Django, …
- Alcuni semplici vincoli
  - Gli utenti non richiedono interi documenti ma fanno richieste frequenti di piccoli file con contenuti puntuali
  - Le Response del Server possono avere tipi differenti
    - text/xml
    - text/plain
    - text/json
    - text/javascript

Sequenza Tipica nella Interazione AJAX

1. Si verifica un evento determinato dall'interazione fra l'utente e la pagina web
2. Si istanzia XMLHttpRequest come risposta all'evento
3. Si configura XMLHttpRequest (si associa una funzione di call-back, si effettua il setup,…)
4. Si effettua una chiamata (tipicamente) asincrona al server
5. Il server elabora la richiesta e risponde al client
6. Si invoca la funzione di call-back che elabora il risultato e normalmente aggiorna il DOM della pagina per visualizzare i risultati della elaborazione

Sequenza Tipica nella Interazione AJAX



Esempio AJAX

- Per vedere il funzionamento di AJAX consideriamo un semplice esempio
- Costruiamo una semplice pagina HTML in cui predisponiamo una form con due campi di testo
  - Name: nome di un giocatore che viene inserito dall'utente
  - Team: il nome della squadra di calcio in cui milita. Il valore viene inserito utilizzando la tecnologia AJAX

Esempio AJAX

```
<html>
<body>
…
<form name="FormEsempio">
Name: <input type="text" name="name" />
Team: <input type="text" name="team"/>
</form>
</body>
</html>
```

Chiaramente manca ancora il componente AJAX che mostreremo nelle prossime slide

Dovremo agganciare il codice AJAX alla form in qualche modo…

Si noti che non è stato inserito alcun bottone per la submit della form

XMLHttpRequest

- L'elemento principale alla base della tecnologia AJAX è l'oggetto XMLHttpRequest
- Esistono varie implementazioni di XMLHttpRequest
  - Internet Explorer utilizza una versione basata su ActiveX
  - Altri browser hanno una differente versione dell'oggetto
  - Il W3C è impegnato in una attività di standardizzazione per cui entro pochi anni i problemi relativi alla eterogeneità dei browser dovrebbero essere risolti
- È necessario instanziare l'oggetto corretto in accordo al browser dell'utente

Esempio AJAX

```
<html>
<body>
<script type="text/javascript">
function firstAjax()
{var xmlHttp;
 try  { xmlHttp=new XMLHttpRequest(); }
catch (e)  {
      try    {xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");}
      catch (e) {try {xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");}
                 catch (e)  {alert("Il tuo browser non è amico di AJAX!");
                             return false;}}}
…. Manca ancora il codice AJAX}
</script>
<form name="FormEsempio">
Name: <input type="text" name="name" />
Team: <input type="text" name="team"/>
</form>
</body>
</html>
```

Se il browser è Firefox, Opera, Safari, Chrome l'operazione va a buon fine altrimenti si solleva una eccezione

Se il browser è IE 6.0+ l'operazione va a buon fine

Si prova con Internet Explorer 5.5+. Se si solleva una eccezione allora si suppone che il browser non supporti AJAX

Manca l'associazione fra la form e la funzione AJAX

Proprietà readystate ed onreadystatechange

- In AJAX si assume che una call-back si faccia carico dell'elaborazione delle risposte che otteniamo dal server

xmlHttp.**onreadystatechange=function()**
{codice della funzione di call-back}

- La proprietà **readystate** mantiene lo **stato** della **risposta** del **server**. Ogni volta che il valore di **readystate** varia, **viene chiamata** la funzione di **call-back** associata ad **onreadystatechange**

I valori possibili di readystate

Valore	*Descrizione*
0	la richiesta non è stata inizializzata
1	la richiesta è stata settata
2	la richiesta è stata spedita
3	la richiesta è in corso di elaborazione
4	la richiesta è stata completata

All'interno della call-back associata ad onreadystatechange dobbiamo quindi andare a discriminare vari casi possibili…

status

- Restituisce lo status della response HTTP
  - 200 OK
  - 404
  - …

reponseText

Handler associato alla richiesta al server

```
…codice browser dependent per istanziare XMLHttpRequest
xmlHttp.onreadystatechange=function(){
        if(xmlHttp.readyState==4)
        {document. FormEsempio.team.value=xmlHttp.responseText;}
                }
…
```

Quando i dati sono arrivati (readystate==4) mettili nel campo team della form FormEsempio

Metodi di XMLHttpRequest

- **open(“HTTP method”, “URL”, syn/asyn)**
  - Metodo GET o POST
  - URL a cui indirizzare la HTTP Request
  - Modalità di interazione (tipicamente asincrona)
- **send(content)**
  - Invia la richiesta includendo (eventualmente) una stringa o dati DOM
- **abort()**
  - Termina la HTTP Request corrente

Metodi di XMLHttpRequest

- **getAllResponseHeaders()**
  - Restituisce gli header come una stringa di coppie attributo/valore
- **getResponseHeader(“header”)**
  - Restituisce il valore di un header specificato
- **setRequestHeader(“label”,”value”)**
  - Consente di settare gli header si richiesta prima della send

Esempio Completo

```
<html>
<body>
<script type="text/javascript">
function firstAjax()
{var xmlHttp;
 try  { xmlHttp=new XMLHttpRequest(); }
catch (e)  {
       try    {xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");}
       catch (e) {try {xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");}
                  catch (e)  {alert("Il tuo browser non è amico di AJAX!");
                                                  return false;}}}
xmlHttp.onreadystatechange=function(){
                              if(xmlHttp.readyState==4)
                              {document. FormEsempio.team.value=xmlHttp.responseText;}
                   }
}
</script>
<form name="FormEsempio">
Name: <input type="text" onkeyup="ajaxFunction();" name="name" />
Team: <input type="text" name="team"/>
</form>
</body>
</html>
```


Altro Esempio : Validazione di una Form

- Vogliamo validare una form utilizzando AJAX
- Non sempre infatti è possibile disporre di tutte le informazioni per la validazione client-side
  - Es. validazione di uno userID

```
<input type="text"
        size="20"
         id="userid"
        name="id"
        onkeyup="validateUserId();">
```

Altro Esempio: Validazione di una Form

```
var req;
function initRequest() {
if (window.XMLHttpRequest) {
   req = new XMLHttpRequest();
} else if (window.ActiveXObject) {
   isIE = true;
  req = new ActiveXObject("Microsoft.XMLHTTP");     } }

function validateUserId() {
initRequest();
req.onreadystatechange = processRequest;
 if (!target) target = document.getElementById("userid");
var url = "validate?id=" + escape(target.value);
req.open("GET", url, true);
req.send(null);    }
```

Abbiamo un po' semplificato

Associa un handler a onreadystatechange

Effettua la chiamata

Altro Esempio: Validazione di una Form

```
function processRequest() {
 if (req.readyState == 4) {
   if (req.status == 200) {
         var message = ...;
  ...
```

Ovviamente dobbiamo aggiornare la pagina utilizzando JavaScript e DOM per visualizzare correttamente i risultati

Questioni Aperte in AJAX

- È accresciuta la complessità delle Web Application
  - La logica di presentazione è ripartita fra client-side e server-side
- Le applicazioni AJAX evidenziano problemi quali il debug, il test ed il mantenimento
  - Il test di codice JavaScript è complesso
  - Il codice JavaScript ha problemi di modularità
- I toolkit AJAX sono molteplici e solo recentemente hanno raggiunto una discreta maturità (es. Mootools, Scriptacolus, Prototype,…)
- Mancanza di standardizzazione di XMLHttpRequest e sua mancanza nei vecchi browser
- Problemi di indicizzazione da parte dei motori di ricerca

AJAX, XML e JSON

responseXML

- Spesso i dati scambiati fra client e server sono codificati in XML
- AJAX come abbiamo visto è in grado di ricevere documenti XML
- In particolare è possibile processare i documenti XML ricevuti utilizzando le API W3C DOM
  - Il modo con cui operiamo su dati in formato XML è analogo a quello che abbiamo visto per ambienti Java
  - Usiamo un parser ed accediamo agli elementi di nostro interesse
  - Per visualizzare i contenuti ricevuti modificchiamo il DOM della pagina HTML

Esempio: responseXML

```
<html>
<head>
<script src="selectmanager_xml.js"></script>
</head>
<body>
<form action=""> Scegli un project manager:
<select name="manager" onchange="showManager(this.value)">
<option value="Carlo11">Carlo Rossi</option>
<option value="Anna23">Anna Bianchi</option>
<option value="Giovanni75">Giovanni Verdi</option>
</select></form>
<b><span id="companyname"></span></b><br />
<span id="contactname"></span><br />
<span id="address"></span>
<span id="city"></span><br/>
<span id="country"></span>
</body>
</html>
```

La form ci consente di scegliere un project manager da una lista

Tramite AJAX visualizziamo i dati del project manager selzionato

Esempio: responseXML

Ipotizziamo che i dati sui project manager siano contenuti in un data-base. Il server riceve una request in cui passiamo un identificativo del project manager, interroga il data-base e ci restituisce un file XML con i dati che abbiamo richiesto. Esempio:

```
<?xml version='1.0' encoding='UTF-16'?>
<company>
<compname>Microsoft</compname>
<contname>Anna Bianchi</contname>
<address>Viale Risorgimento 2</address>
<city>Bologna</city>
<country>Italy</country>
</company>
```


Esempio: responseXML

selectmanager_xml.js

```
var xmlHttp;
function showManager(str)
{
xmlHttp=GetXmlHttpObject();
if (xmlHttp==null)
  { alert ("Il tuo browser non è amico di AJAX!");
  return;}
var url="getmanager_xml.jsp";
url=url+"?q="+str;
xmlHttp.onreadystatechange=stateChanged;
xmlHttp.open("GET",url,true);
xmlHttp.send(null);
}
```

Funzione (definita dopo) che restituisce un'istanza di XMLHttpRequest o null se il browser non supporta AJAX

URL del server e parametri per la query (identificatore del poject manager)

Associo un handler per la call-back onreadystatechange

Request asincrona di tipo GET

Esempio: responseXML

selectmanager_xml.js

```
function GetXmlHttpObject()
{var xmlHttp=null;
 try
  { xmlHttp=new XMLHttpRequest(); } // Firefox, Opera 8.0+, Safari
 catch (e){ // Prova internet explorer
    try {xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");}
    catch (e) {xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");}
  }
return xmlHttp;
}
```


Esempio: responseXML

selectmanager_xml.js

```
function stateChanged()
{
if (xmlHttp.readyState==4)
{
var xmlDoc=xmlHttp.responseXML.documentElement;

document.getElementById("companyname").innerHTML=xmlDoc.getElem
entsByTagName("compname")[0].childNodes[0].nodeValue;

… analogamente per gli altri campi da riempire…
}
}
```

Come evidente si usa DOM per aggiornare la pagina con i dati acceduti sempre tramite DOM nel documento XML ricevuto

JavaScript Object Notation (JSON)

- JSON è un formato per lo scambio di dati molto più leggero di XML
- JSON è molto semplice sia per la lettura/scrittura da parte degli sviluppatori, sia per l'elaborazione da parte delle macchine
- JSON è un linguaggio testuale largamente supportato dai maggiori linguaggi di programmazione

JSON

- Gli oggetti JSON sono dotati di tipo
  - Stringhe, array, numeri
- È particolarmente adatto a lavorare con JavaScript
  - I dati sono accessibili direttamente in JavaScript e non è richiesto il parsing (anche se è fortemente consigliato qualora ci siano problemi di sicurezza)
  - L'accesso ai dati equivale all'accesso alle proprietà di un oggetto JavaScript

Struttura delle implementazioni di JSON

- JSON è stato definito sulla base di due strutture fondamentali
  - Una collezione di coppie attributo/valore
  - Una lista ordinata di valori
- I linguaggi che supportano JSON forniscono tipicamente un supporto per queste strutture e possono relizzarle come
  - Oggetti, dizionari, tavole hash
  - Array, vettori, liste, …

JSON Object Notation

- Un oggetto JSON è un insieme (non ordinato) di coppie attributo/valore
- Gli oggetti JSON sono racchiusi fra parentesi graffe
- Ogni attributo è seguito dai due punti “:” ed è separato dalle altre coppie da una virgola “,”

Esempio di JSON

```
var myJSONObject = {"bindings": [
{"ircEvent": "PRIVMSG", "method": "newURI", "regex": "^http://.*"},
{"ircEvent": "PRIVMSG", "method": "deleteURI", "regex": "^delete.*"},
{"ircEvent": "PRIVMSG", "method": "randomURI", "regex": "^random.*"}
 ] };
```

Abbiamo definito un oggetto che contiene un solo membro “bindings” associato ad un array “[…]” che contiene tre oggetti. Come normalmente avviene per gli array gli elementi sono tre oggetti omogenei che contengono loro volta tre membri: “ircEvent”, “method” e “regex”

I membri possono essere riferiti usando la notazione consueta che utilizziamo in vari linguaggi di programmazione (sia per l’accesso agli array sia per l’accesso ad attributi pubblici degli oggetti). Esempio:

myJSONObject.bindings[0].method

Conversione fra Testo ed Oggetto JSON

- In JavaScript per convertire un documento in formato JSON in un oggetto possiamo utilizzare **eval()**
  - JSON è un sotto-insieme di JavaScript ed eval si occupa di effettuare il parsing del testo e di istanziare un oggetto JavaScript a cui possiamo facilmente accedere

var myObject = **eval('(' + myJSONtext + ')')**;

Parser JSON

- Possiamo avere dei problemi relativi alla sicurezza utilizzando direttamente il comando eval()
  - Usare perciò eval solo quando la sorgente del codice JSON è fidata (cosa che normalmente possiamo assumere), viceversa utilizzare un parser JSON

var myObject = myJSONtext.parseJSON();

Conversione di un Oggetto in JSON

var myJSONText = myObject.toJSONString();

- È anche possibile convertire strutture dati JavaScript in JSON
- Limitazione: JSON non supporta le strutture cicliche

JSON e AJAX

I dati in formato JSON sono ricevuti dal client come una stringa. Con il comando eval() possiamo istanziare un oggetto JavaScript ed accedere ai suoi attributi utilizzando la solita sintassi per l'accesso agli array e la dot-notation per l'accesso agli attributi. Esempio:

```
var JSONdata = eval(req.responseText);
var name = JSONdata.name;
var address = JSONdata.addresses[3];
var streetname = JSONdata.addresses[3].street;
```


Inviare una HTTP Request JSON

Possiamo anche volere mandare una request al server in formato JSON

1. Creare un oggetto JavaScript JSON
2. Settare la XMLHttpRequest
3. Predisporre un invio di tipo "POST" della Request
4. Passare l'oggetto JSON nella send

Esempio di Request JSON

```
var carAsJSON = JSON.stringify(car);
var url = "JSONExample?timeStamp=" + new Date().getTime();
createXMLHttpRequest();
xmlHttp.open("POST", url, true);
xmlHttp.onreadystatechange = handleStateChange;
xmlHttp.setRequestHeader("Content-Type",
    "application/x-www-form-urlencoded");
xmlHttp.send(carAsJSON);
```