javascript - Ajax

Andrea Ferracani
Comunicazione client - server
andreaferracani@gmail.com

UNIVERSITÀ DEGLI STUDI FIRENZE
SIAF SISTEMA INFORMATICO DELL'ATENEO FIORENTINO

DOM - Document Object Model

Il **Document Object Model** (**DOM**) è una convenzione cross-platform e indipendente dal linguaggio per rappresentare ed interagire con gli oggetti in documenti HTML, XHTML, XML.

Gli oggetti costituenti l'albero del DOM possono essere chiamati e manipolati attraverso l'uso di metodi applicabili agli oggetti stessi.

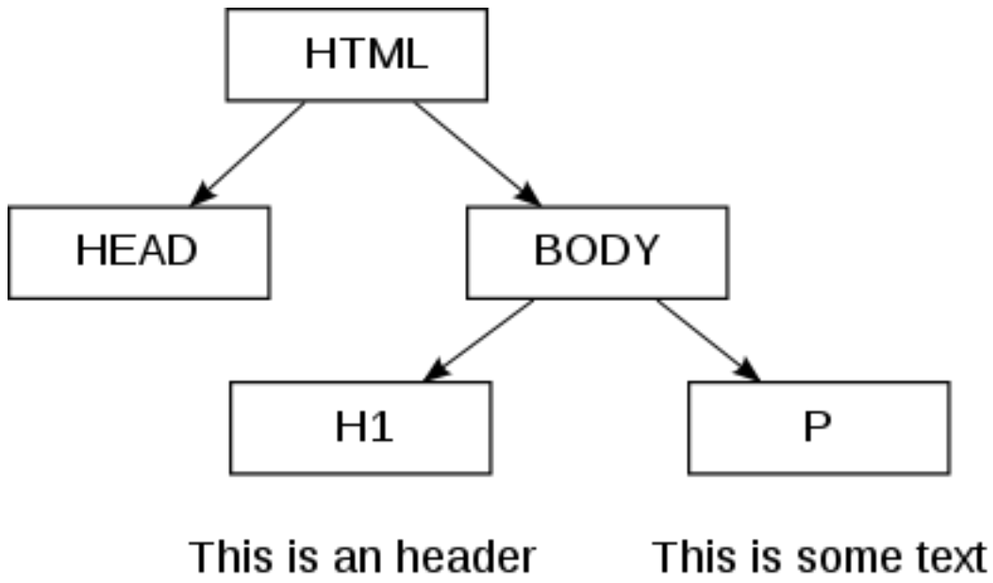
L'interfaccia pubblica del DOM è specificata nella cosiddetta **Application Programming Interface** (API).

I Web browsers usually usano solitamente un modello assimilabile al DOM per visualizzare e gestire una pagina web (per esempio una pagina HTML). **Le DOM APIs sono inoltre usate per interagire con una pagina Web via Javascript code**. In altre parole, il Document Object Model è il modo in cui JavaScript vede lo stato del browser e la pagina HTML che vi è contenuta.

Quando una pagina HTML viene visualizzata nel browser, il browser decodifica (parsa) il markup (HTML), che è stato scaricato dal web-server, e lo converte in un **in-memory DOM**. Il DOM è usato per generare altre strutture che consentono di visualizzare la pagina nella finestra del browser.

I nodi di ogni documento DOM sono organizzati in una **tree structure, chiamata il DOM tree**. Il nodo principale nel DOM tree è il **Document** object. Ogno nodo ha zero o più figli.

DOM - Document Object Model




```
|-> Document
  |-> Element (<html>)
    |-> Element (<body>)
      |-> Element (<div>)
        |-> text node
        |-> Anchor
          |-> text node
|-> Form
        |-> Text-box
        |-> Text Area
        |-> Radio Button
        |-> Check Box
        |-> Select
        |-> Button
```

What’s AJAX

Ajax è una tecnologia che mira a **migliorare la user experience** di una applicazione web.

L’acronimo sta per **Asynchronous JavaScript and XML**.

Ajax non rappresenta una nuova tecnologia. Si tratta di **un insieme di tecnologie**:

- **Javascript**: serve a creare le funzionalità client-side, per manipolare gli oggetti del DOM (document object model)
- **XMLHttpRequest**: è l’oggetto che consente la comunicazione asincrona con il server. La comunicazione avviene attraverso l’invio di parametri (coppie nome / valore) in GET o POST.
- **Un linguaggio server side**: per esempio PHP, che non risponderà una pagina HTML ma attraverso un linguaggio di interscambio dati: **XML** o **JSON** (JavaScript Object Notation)

  Javascript intro: http://www.w3schools.com/js/js_intro.asp, http://www.w3schools.com/jsref/default.asp
  XMLHttpRequest: http://www.w3schools.com/xml/xml_http.asp
  XML and JSON samples: http://www.w3schools.com/xml/xml_view.asp, http://www.w3schools.com/json/default.asp
  JSON intro: http://www.w3schools.com/json/default.asp

Javascript - intro

JavaScript è un linguaggio di **scripting**, cioè è un linguaggio di programmazione interpretato, JavaScript può essere utilizzato per controllare quasi tutte le componenti del browser e per rispondere a varie azioni dell'utente, quali l'immissione nei moduli e la navigazione nella pagina.

È particolarmente utile perché tutti i compiti di elaborazione sono scritti nello script (incorporato nel documento HTML), e quindi l'intero processo definito dallo script è eseguito lato client (cioè all'interno del browser), senza la necessità di fare riferimento a un server.

In altri termini, invece di fare eseguire al server dei compiti e fornire i risultati al browser, quest'ultimo può gestire localmente alcuni compiti, dando così all'utente tempi di risposta più brevi e riducendo il traffico di rete.

Quando un documento HTML con uno script di JavaScript è caricato in un browser che supporta questo linguaggio, le funzioni dello script vengono calcolate, memorizzate, ed eseguite quando si verificano determinati **eventi** (ad esempio quando l'utente sposta il mouse sopra un oggetto o immette un testo in una casella).

Javascript - storia

Nel **1995** Netscape decise di dotare il proprio browser di un linguaggio di scripting che permettesse ai web designer di interagire con i diversi oggetti della pagina (immagini, form, link, ecc.), ma soprattutto con le applet Java (Netscape Navigator 2.0: LiveScript).

Microsoft rispose a JavaScript in due modi:

- con l'introduzione di VBScript all'interno di **Internet Explorer 3**
- con una propria versione di JavaScript, sotto molti aspetti simile all'originale, chiamata **JScript** (siamo nel luglio 1996)

A causa di alcune differenze presenti in Internet Explorer 3 Netscape e Sun decisero di standardizzare JavaScript e si affidano all'**European Computer Manufacturers Association (ECMA)**.

ECMAScript è dunque figlio di JavaScript. E oggi quando si dice JavaScript, JScript ed ECMAscript sostanzialmente si indicano tre varietà dello stesso linguaggio.

Bisogna poi tener conto che differenti versioni del browser, implementano differenti versioni di JavaScript (la più recente è la 1.4, mentre la 1.5 è ancora in beta), quindi il modo di interpretare determinati costrutti potrebbe variare da una sottoversione del browser all'altra.
(Le future versioni saranno Javascript 2.0 ed ECMAScript 4.0)

Javascript - DHTML

Una pagina “dinamica” (**DHTML**) è una pagina HTML contenente codice JavaScript associato a determinati eventi che implementa specifiche funzionalità.
Non bisogna confondere una pagina DHTML con una pagina generata dinamicamente dal server (ASP, PHP, JSP, etc).

JavaScript è attualmente il linguaggio di programmazione **più popolare**.
Viene usato per l’HTML, il Web, lato client e server, sui tablet, gli smartphone, and più.

Si può usare javascript per:

- cambiare elementi HTML
- distruggere elementi HTML
- creare nuovi elementi HTML
- copiare e clonare elementi HTML
- e tanto altro ...

Javascript vs Java

JavaScript non va confuso con Java, dato che tra i due linguaggi esistono importanti differenze:

JavaScript	***Java***
è un linguaggio interpretato	è un linguaggio compilato
viene eseguito sul client	viene eseguito sul server
non consente di scrivere programmi autonomi	consente di scrivere programmi autonomi

Javascript

Per inserire codice JavaScript in un documento HTML esistono tre modi:

- **elemento <script>** contenente il codice JavaScript:
  <script type=“text/javascript”></script>

- inclusione di **script esterni**
  <script src=“script_esterno.js"></script>

- **direttamente nel codice** HTML in risposta ad eventi
  <span onclick=“alert(‘Ciao Mondo!’);”>Clicca qui

Esercizi su funzioni di base

Esercizio 1 - getElementById
Esercizio 2 - inline javascript
Esercizio 3 - javascript di pagina
Esercizio 4 - cambia attributo
Esercizio 5 - gestore d'evento
Esercizio 6 - creazione dinamica elemento e gestione eventi
Esercizio 7 - evento tastiera

Advanced staff

Esercizio 8 - trascinare
Esercizio 9 - disegnare
Esercizio 10 - disegnare su mobile

BOM - Browser Object Model

Il **Browser Object Model (BOM)** è una convenzione specifica dei browser che si riferisce a tutti gli oggetti esposti. Diversamente dal **Document Object Model**, non esiste uno standard di riferimento da osservare ed i venditori di browser sono liberi di implementarlo come credono.

- L'oggetto **window** è supportato da tutti i browser, di cui rappresenta la finestra. Tutti gli oggetti, le funzioni e le variabili javascript diventano automaticamente membri e proprietà di questo oggetto. Esempio

  window.document.getElementById("header");

  è lo stesso di:

  document.getElementById("header");

- L'oggetto **window.screen** contiene informazioni riguardanti lo schermo del computer dell'utente. Esempio

Per esempio se volessi stampare la larghezza dello schermo "ScreenWidth: " + **screen.width**, avrebbe come risultato: Screen Width: 1440

BOM - Browser Object Model

- L’oggetto **navigator** contiene informazioni relative al browser. Esempio

  Per esempio la variabile:

  var nomeBrowser = "Nome Browser: " + navigator.appCodeName;

  avrà come risultato se stampata a schermo “Nome Browser: Mozilla”

- L’oggetto **history** contiene le URLs visitate dall'utente (dentro una finestra del browser). E’ parte dell’oggetto windows può essere chiamato attraverso la proprietà window.history. Esempio

  Ha solo una proprietà [length] e tre metodi [back, forward, go]

- L’oggetto **location** contiene informazioni sulla URL corrente. E’ parte dell’oggetto windows può essere chiamato attraverso la proprietà window.location. Esercizio

Perché AJAX

Elimina il page reload della pagina: è in grado di fare chiamate asincrone ad un server.

Scenario

Se dovessimo implementare **un sistema di suggerimento per una form di input** avremmo 3 possibili scenari:

- L'utente riempie e sottomette la form (html) al server (php per esempio). Il server controlla i dati e invia la risposta. In questo caso fra richiesta e risposta esiste un **time delay**.
- La richiesta viene fatta **usando JavaScript** mentre l'utente sta scrivendo. In questo caso non c'è ritardo temporale ma **si sfruttano un insieme di funzionalità** del linguaggio client-side.
- **Attraverso Ajax**, si usa ancora JavaScript, ma è possibile comunicare con il server mentre l'utente sta scrivendo nei campi della form

AJAX è stato reso popolare nel 2005 da Google, con **Google Suggest**.

Google Suggest usa AJAX per creare una interfaccia web dinamica: quando si comincia a scrivere nel Google Search Box, JavaScript invia le lettere al server e il server restituisce un insieme di suggerimenti.

Altre tecnologie client side

Sul tuo computer si possono anche usare:

Java applets: girano nel browser attraverso la Java virtual machine
Macromedia Flash: gira nel browser con il Flash player plugin
Microsoft Silverlight: gira nel browser attraverso una versione leggera del NETFramework

Queste tecnologie sono piuttosto **pesanti per tasks semplici**, quali il suggerimento nella input di una form.

AJAX history

1995: caricamento asincrono disponibile nelle Java applets nella prima versione del linguaggio Java.
1996: Internet Explorer introduce l'elemento iframe all'HTML, che consente il loading asincrono.
1999: Microsoft utilizza la tecnologia iframe per cambiare dinamicamente le news e gli stock quotes nella homepage di Internet Explorer (http://home.microsoft.com).
1999, Microsoft crea il controllo XMLHTTP ActiveX in Internet Explorer 5, adottato poi da Mozilla, Safari, Opera e altri browser come l'XMLHttpRequest JavaScript object.
Microsoft ha adottato l'oggetto XMLHttpRequest con Internet Explorer 7, anche se la versione ActiveX è ancora supportata.

Best of AJAX

There are many Ajax applications, and we use it every day!

- Bing Maps
- Google Maps
- Yahoo Maps
- Flickr
- Picasa Web Albums
- Google
- Yahoo
- Gmail
- Digg

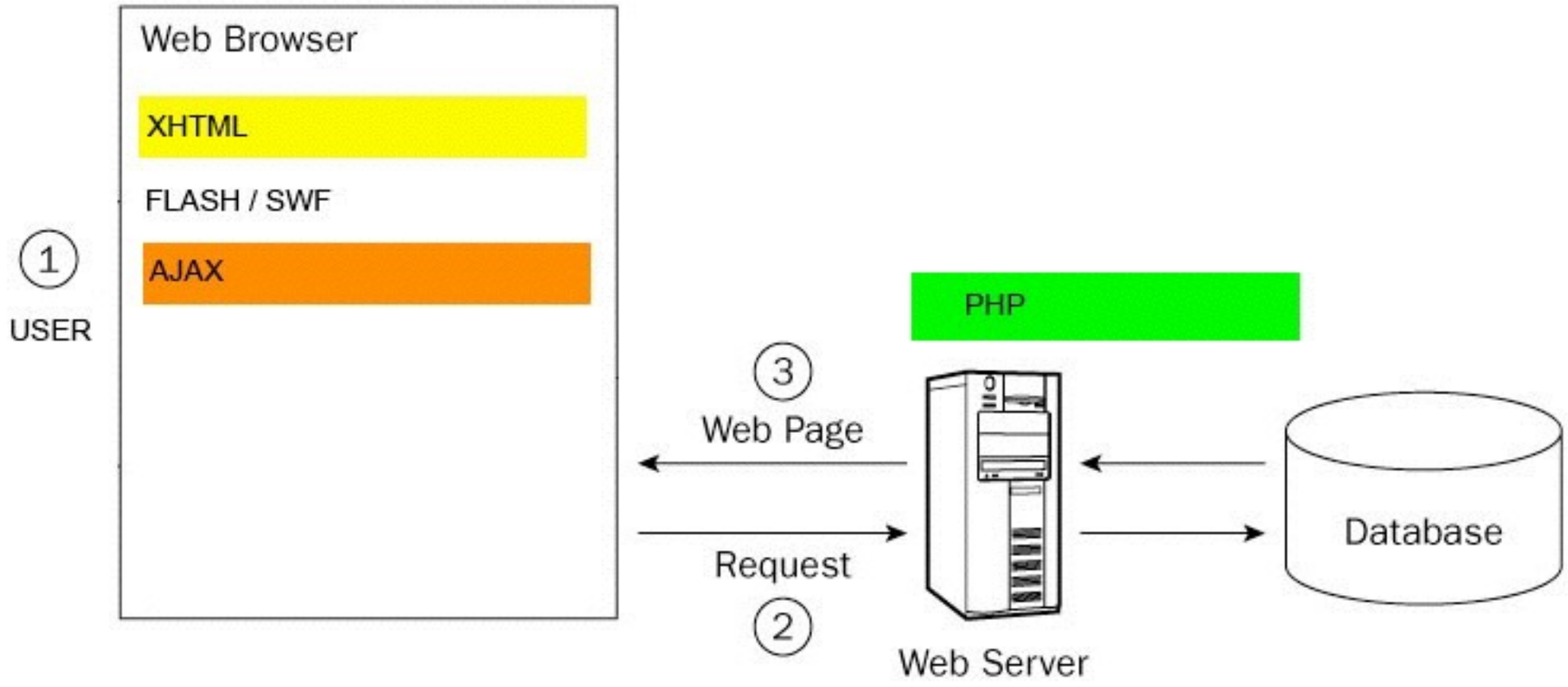
Benefits vs. disavantages

Benefits:

- si possono creare applicazioni **responsive ed intuitive**
- usa tecnologie esistenti dunque **non c'è niente da installare**

Svantaggi:

- le persone spesso dimenticano che **non tutti i browser supportano JavaScript** o che può essere disabilitato client-side: le funzionalità di base dell'applicazione dovrebbero comunque essere supportate. **Ajax is a plus**.
- I motori di ricerca non sono in grado di eseguire JavaScript. Perciò **i contenuti caricati con Ajax** non sono indicizzati. Il **bookmark** di una pagina AJAX è problematico perché l'indirizzo sulla barra degli indirizzi non cambia al cambiare dei contenuti. E@ necessario salvare gli stati dell'applicazione, per esempio attraverso le ancore. www.my-example.com/my-Ajax-app/#state1
- Stesso problema del punto precedente riguarda **i bottoni di back and forward**.
- Same origin policy impedisce ad alcune tecniche Ajax di essere usate fra domini diversi, sebbene il W3C abbia stilato un draft su XMLHttpRequest object che lo dovrebbe consentire. Esistono delle alternative come l'uso di un Cross Domain Communications channel in un iframe dentro la pagina, o l'uso di un protocollo come JSONP.

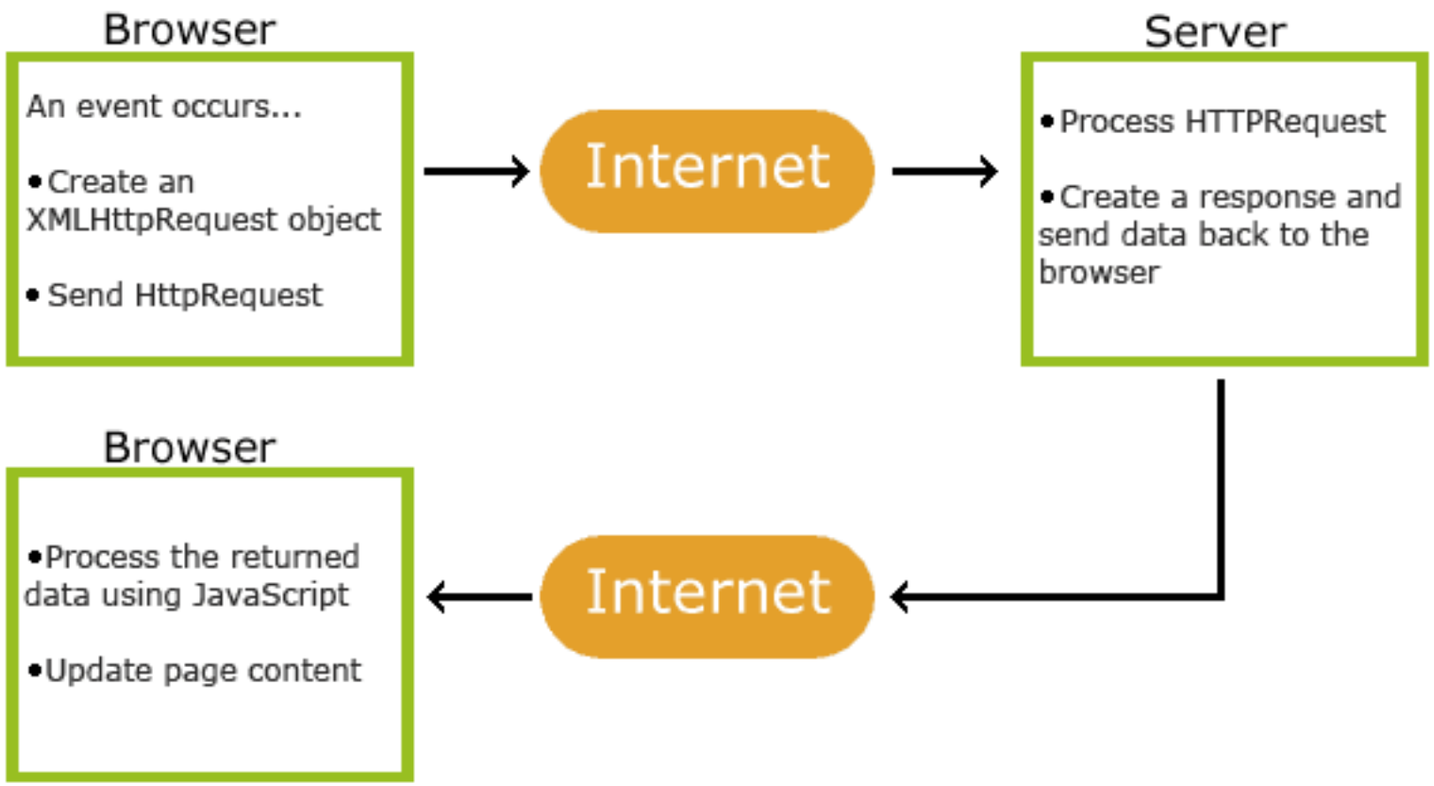


- XHTML: definisce il markup di una pagina web
- FLASH: aggiunge interattività alla pagina HTML e comunica con PHP
- AJAX: aggiunge interattività alla pagina HTML e comunica con PHP
-

- PHP: comunica con FLASH /AJAX widgets e con il database

AJAX in short

- Invia richieste al server attraverso **XMLHttpRequest object**
- Gestisce le risposte in **XML** o **JSON**
- Può creare **riferimenti a tags** nella pagina HTML
- **Genera HTML** parsando le richieste ricevute dal server



XMLHttpRequest - First sample

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "
http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">

<head>
<link rel="stylesheet" href="../../css/style.css" type="text/css" />

        <!-- CHIAMATA A JAVASCRIPT -->
        <script type="text/javascript" src="json2.js"></script>
        <script type="text/javascript" src="mysql_3.js"></script>

</head>
```

XMLHttpRequest - sample

http://localhost:8888/ajax_course_2011/examples/5_mysql/index.html

Loads a list of books asynchronously

XHTML code

```
//function called when the page loads
<body>

        <div class="content">

                <h2>PHP e MYSQL</h2>
                <p>Questa lista di libri è caricata dinamicamente dal <strong>database</strong> con
                <strong>AJAX</strong>!</p>

                //function called when user press the button
                <button onclick='process()'>Carica lista</button>

                 <!-- wrapper of server response -->
                 <div id="myDivElement"></div>

        </div>

    </body>
</html>
```

XMLHttpRequest - sample

AJAX code

```
// contiene il riferimento all'oggetto XMLHttpRequest
var xmlHttp = createXmlHttpRequestObject();

// funzione cross browser per creare l'istanza
function createXmlHttpRequestObject() {

    // variabile che contiene il riferimento locale all' XMLHttpRequest object
    var xmlHttp;

    // se il browser è Internet Explorer 6 o più vecchio
    if(window.ActiveXObject) {
        try {
            xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
        } catch (e) {
            xmlHttp = false;
        }
    }
```

XMLHttpRequest - sample

Codice AJAX

```
// controlla se Mozilla o altri
else {
        try {
                xmlHttp = new XMLHttpRequest();
        } catch (e) {
                xmlHttp = false;
        }
}
        // controlla se l’oggetto esiste o ritorna l’errore
        if (!xmlHttp) {
                alert(“Errore nella creazione dell’oggetto XMLHttpRequest");
        } else {
                return xmlHttp;
        }
}
```

XMLHttpRequest - sample

Codice AJAX

```
// funzione che apre la connessione con PHP per mezzo di XMLHttpRequest e mette l'oggetto in attesa della risposta

function process() {
        // se XMLHttpRequest esiste
        if (xmlHttp) {

        // prova a connetterti al server
        try {
            // apri la connessione
            xmlHttp.open("GET", "index.php", true);

            // setta il gestore d'evento
            xmlHttp.onreadystatechange = handleRequestStateChange;

            // invia la richiesta
            xmlHttp.send(null);
        }
        // cattura l'errore
        catch (e) {
            alert("Impossibile connettersi al server:\n" + e.toString());
        }
}
```

XMLHttpRequest - sample

AJAX code

```
// funzione che controlla lo stato della risposta del server nella variabile readyState

function handleRequestStateChange() {

    // se readyState è 4 la risposta è completata
    if (xmlHttp.readyState == 4) {

        // controlla se lo stato della risposta HTTO è ok
        if (xmlHttp.status == 200) {
            try {
                // se OK, fai qualcosa
                handleServerResponse();
            } catch(e) {
                // mostra il messaggio di errore
                alert(“Errore nel ricevere la risposta HTTP: " + e.toString());
            }
        } else {
            // show status message
            alert("Errore nel ricevere la risposta:\n" + xmlHttp.statusText);
        }
    }
}
```

XMLHttpRequest - sample

AJAX code

```
// funzione che gestisce la risposta del server

function handleServerResponse() {

    // leggi la risposta come JSON
    responseJSON = JSON.parse(xmlHttp.responseText);

    // genera una variabile che contenga l'output html
    var html = "<ul>";

    // itera l'array
    for (var i=0; i<responseJSON.books.length; i++) {
        html += "<li>" responseJSON.books[i].title +  "</li>";
    }
    html += "<li>";

    // ottieni un riferimento alla ad un elemento della pagina HTML
    myDiv = document.getElementById("myDivElement");

    // genera HTML
    myDiv.innerHTML = "<p>Server says: </p>" + html;
}
```

XMLHttpRequest - sample

PHP code

//**index.php** - questo file fa le query al database, formatta la risposta e la restituisce al client

```
<?php
  // aggiungi gli header alla risposta che comunichino il tipo
  header('Content-Type: text/json');

   // includi file necessari per accesso DB e debug
  require_once('config.php');
  require_once('error_handler.php');

// apri connessione al database
$mysqli = new mysqli(DB_HOST, DB_USER, DB_PASSWORD, DB_DATABASE);

// formula la query
$query = 'SELECT book_id, book_title FROM books';

// esegui la query
$result = $mysqli->query($query);

// crea un array per contenere l'elenco di libri
$books = array();
```

XMLHttpRequest - sample

PHP code

```
// scorri il risultato
while ($row = $result->fetch_array(MYSQLI_ASSOC)) {

    // estrai id e titolo
    $book_id = $row['book_id'];
    $book_title = $row['book_title'];

    // crea un array associativo per ogni record e mettilo nel contenitore
    $book = array('title' => $book_title,'id' =>$book_id);
    array_push($books, $book);
}

// crea un nuovo contenitore per definire la chiave principale
$response = array('books' => $books);

// converti a JSON
echo json_encode($response);

// chiudi l'input stream
$result->close();

// chiudi la connessione al DB
$mysqli->close();
?>
```

XMLHttpRequest - sample

PHP code

```
// Il File config.php: definisci le costanti di accesso al database

<?php

    define('DB_HOST', 'localhost');

    define('DB_USER', 'ajax_course');

    define('DB_PASSWORD', 'ajax_course');

    define('DB_DATABASE', 'ajax_course');
?>
```

XMLHttpRequest - sample

Codice PHP

```
// File error_handler.php: override della funzione di default per la gestione degli errori

<?php
   // assign default action for error handling
   set_error_handler('error_handler', E_ALL);

   // definisci la nuova funzione
   function error_handler($errNo, $errStr, $errFile, $errLine) {

      // pulisci l'output già generato
      if(ob_get_length()) ob_clean();

      // output
      $error_message = 'ERRNO: ' . $errNo.chr(10) . 'TEXT: ' . $errStr.chr(10) . 'LOCATION: ' . $errFile .
                       ', line ' . $errLine;

      echo $error_message;
       exit;
  }
?>
```

XMLHttpRequest - Altri esempi

- http://localhost:8888/ajax_course_2014/examples.html
- http://www.w3schools.com/ajax/ajax_aspphp.asp
- http://www.w3schools.com/ajax/ajax_database.asp
- http://www.w3schools.com/ajax/ajax_xmlfile.asp

RDBMS

Abbiamo usato un **Database Management System (RDBMS)** per storare i dati. (MAMP / XAMP / WAMP / LAMP / AppServ)

Si deve:

- Creare un **database**
- Scrivere **SQL queries** per trovare e manipolare i dati
- Connettersi a MySQL attraverso **PHP code**
- **Mandare SQL queries** al database

Quando si creano le tabelle in un database il significato di questi termini è importante:

- **Primary keys**: la primary key è una colonna speciale che assicura che ciascun record sia unico (usualmente la colonna ID)
- **Data types**: tipi numerici, caratteri e stringhe, date
- **NULL** e **NOT NULL**: se il valore di una cella di una colonna può essere vuoto o meno
- **Default values**: valori di default delle colonne
- **Auto_increment**: colonne che si autoincrementano, di solito usato per la chiave primaria
- **Indexes**: sono oggetti del database che velocizzano la ricerca, sfortunatamente rallentano le altre operazioni!

RDBMS

SELECT esempio

SELECT <column list>
FROM <table name(s)>
[WHERE <restrictive condition(s)>]

INSERT esempio

INSERT INTO <table name> [(column list)]
VALUES (column values)

UPDATE esempio

UPDATE <table name>
SET <column name> = <new value> [, <column name> = <new value> ...]
[WHERE <restrictive condition>]

DELETE esempio

DELETE FROM <table name>
[WHERE <restrictive condition>]

Aggiungere record nel nostro database:

INSERT INTO books (book_title) VALUES ('Ajax first book');
INSERT INTO books (book_title) VALUES ('Ajax second book');
INSERT INTO books (book_title) VALUES ('Ajax third book');

XHR2 e HTML5

Strettamente parlando **XHR2 non è HTML5**. Comunque è parte delle migliorie che i venditori di browser sta aggiungendo alla specifica.

XMLHttpRequest Level 2 introduce nuove possibilità:

- richieste cross-domain,
- eventi di upload progress
- supporto per uploading/downloading binary data.

Ciò consente ad AJAX di lavorare in sincro con altre nuove features dell'HTML5 APIs come **File System API**, Web Audio API, e WebGL.

XHR2 support: http://caniuse.com/xhr2

XHR2 and HTML5

Adesso XHR può specificare un formato di risposta:

xhr.**responseType**

Prima di inviare una richiesta, setta xhr.responseType a "text", "arraybuffer", "blob", o "document", in base ai bisogni. Settare xhr.responseType = '' (o omettere il parametro) metterà di default "text".

xhr.**response**

Dopo una richiesta andata a buon fine, l'oggetto xhr conterrà la risposta come una DOMString, ArrayBuffer, Blob, or Document (in base a cosa si è dettato su xhr.responseType)

```
var xhr = new XMLHttpRequest();
xhr.open('GET', '/path/to/image.png', true);
xhr.responseType = 'blob';

xhr.onload = function(e) {
  if (this.status == 200) {
    // Note: .response instead of .responseText
    var blob = new Blob([this.response], {type: 'image/png'});
    ...
  }
};

xhr.send();
```

XHR2 download dati

Un blob può essere usato in diversi modi, incluso salvarlo in **indexedDB**, scriverlo via **HTML5 File System**, o creando una **Blob URL**, come visto in questo esempio.

```
window.URL = window.URL || window.webkitURL;  // Take care of vendor prefixes.

var xhr = new XMLHttpRequest();
xhr.open('GET', '/path/to/image.png', true);
xhr.responseType = 'blob';

xhr.onload = function(e) {
  if (this.status == 200) {
    var blob = this.response;

    var img = document.createElement('img');
    img.onload = function(e) {
      window.URL.revokeObjectURL(img.src); // Clean up after yourself.
    };
    img.src = window.URL.createObjectURL(blob);
    document.body.appendChild(img);
    ...
  }
};

xhr.send();
```

XHR2 data upload

Caricare dati in diversi formati può essere utile, ma non sarebbe utile se non potessimo anche inviarli al server.

XMLHttpRequest aveva finora il limite di poter inviare DOMString or Document (XML) data. **Non più**.

A revamped **send() method** has been overridden to accept any of the **following types**:

- DOMString
- Document
- FormData
- Blob
- File
- ArrayBuffer.

XHR2 data upload

Inviare stringhe: xhr.send(DOMString)

```
function sendText(txt) {
  var xhr = new XMLHttpRequest();
  xhr.open('POST', '/server', true);
  xhr.onload = function(e) {
    if (this.status == 200) {
      console.log(this.responseText);
    }
  };
  xhr.send(txt);
}

sendText('test string');
```

```
function sendTextNew(txt) {
  var xhr = new XMLHttpRequest();
  xhr.open('POST', '/server', true);
  xhr.responseType = 'text';
  xhr.onload = function(e) {
    if (this.status == 200) {
      console.log(this.response);
    }
  };
  xhr.send(txt);
}

sendTextNew('test string');
```

XHR2 data upload

Molte persone sono abituate ad usare **jQuery plugins** o altre librerie per gestire l'invio di form ajax. Adesso, possiamo usare **FormData**, un altro tipo di dato concepito per XHR2. FormData è utile per creare <form> on-the-fly, in JavaScript.

```
function sendForm() {
  var formData = new FormData();
  formData.append('username', 'johndoe');
  formData.append('id', 123456);

  var xhr = new XMLHttpRequest();
  xhr.open('POST', '/server', true);
  xhr.onload = function(e) { ... };

  xhr.send(formData);
}
```

XHR2 data upload

Gli oggetti FormData possono essere inizializzati a partire da un pre-esistente **HTMLFormElement** nella pagina.

```
<form id="myform" name="myform" action="/server">
  <input type="text" name="username" value="johndoe">
  <input type="number" name="id" value="123456">
  <input type="submit" onclick="return sendForm(this.form);">
</form>
```

```
function sendForm(form) {
  var formData = new FormData(form);

  formData.append('secret_token', '1234567890'); // Append extra data before send.

  var xhr = new XMLHttpRequest();
  xhr.open('POST', form.action, true);
  xhr.onload = function(e) { ... };

  xhr.send(formData);

  return false; // Prevent page from submitting.
}
```

XHR2 data upload - file

Una form HTML pup includere **file uploads** (e.g. <input type="file">) e FormData può gestire anche questo. Si appende il file ed il browser si occupa di costruire **richieste multipart/form-data** quando send() viene chiamato:

```
function uploadFiles(url, files) {
  var formData = new FormData();

  for (var i = 0, file; file = files[i]; ++i) {
    formData.append(file.name, file);
  }

  var xhr = new XMLHttpRequest();
  xhr.open('POST', url, true);
  xhr.onload = function(e) { ... };

  xhr.send(formData);  // multipart/form-data
}

document.querySelector('input[type="file"]').addEventListener('change',
function(e) {
  uploadFiles('/server', this.files);
}, false)
```

XHR2 data upload

Si possono anche inviare **File o Blob data** con XHR. Files sono Blobs, quindi funziona allo stesso modo.

```
<progress min="0" max="100" value="0">0% complete</progress>
```

```
function upload(blobOrFile) {
  var xhr = new XMLHttpRequest();
  xhr.open('POST', '/server', true);
  xhr.onload = function(e) { ... };

  // Listen to the upload progress.
  var progressBar = document.querySelector('progress');
  xhr.upload.onprogress = function(e) {
    if (e.lengthComputable) {
      progressBar.value = (e.loaded / e.total) * 100;
      progressBar.textContent = progressBar.value; // Fallback for unsupported browsers.
    }
  };

  xhr.send(blobOrFile);
}
```

XHR2 data upload

Inoltre si possono usare **chunks of data** con **ArrayBuffers**

```
function sendArrayBuffer() {
  var xhr = new XMLHttpRequest();
  xhr.open('POST', '/server', true);
  xhr.onload = function(e) { ... };

  var uInt8Array = new Uint8Array([1, 2, 3]);

  xhr.send(uInt8Array.buffer);
}
```

XHR2 - Cross Origin Resource Sharing (CORS)

CORS permette alle applicazioni web in un dominio di fare chiamate AJAX ad un altro dominio. E' semplice da abilitare, richiedendo che un solo response header sia restituito dal server.

Poniamo di avere un applicazione che gira su **example.com** e volere ottenere dei dati da **www.example2.com**. Normalmente la richiesta fallirebbe ed il browser restituirebbe errore. Con CORS, www.example2.com can può abilitare le richieste con una sola riga

```
Access-Control-Allow-Origin: http://example.com
```

```
Access-Control-Allow-Origin: *
```

Come abilitare cors: http://enable-cors.org/

```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'http://www.example2.com/hello.json');
xhr.onload = function(e) {
  var data = JSON.parse(this.response);
  ...
}
xhr.send();
```

XHR2 - Pratica 1

Mettiamo di avere una **galleria di immagini** e di volere recuperare alcune immagini e **salvarle localmente** usando **HTML5 File System**. Un modo potrebbe essere recuperare le immagini come **Blobs** e scriverle usando **FileWriter**.

```
window.requestFileSystem  = window.requestFileSystem || window.webkitRequestFileSystem;

function onError(e) {
  console.log('Error', e);
}

var xhr = new XMLHttpRequest();
xhr.open('GET', '/path/to/image.png', true);
xhr.responseType = 'blob';

xhr.onload = function(e) {

  window.requestFileSystem(TEMPORARY, 1024 * 1024, function(fs) {
    fs.root.getFile('image.png', {create: true}, function(fileEntry) {
      fileEntry.createWriter(function(writer) {
        writer.onwrite = function(e) { ... };
        writer.onerror = function(e) { ... };
        var blob = new Blob([xhr.response], {type: 'image/png'});
        writer.write(blob);
      }, onError);
    }, onError);
  }, onError);
};

xhr.send();
```

XHR2 - Practice 2

Usando File APIs, possiamo minimizzare il lavoro **per upload are un file di grandi dimensioni**. La tecnica consiste di dividerla in **multiple chunks**, creare un XHR per ciascuno di essi, e rimettere assieme il file sul server. Il procedimento è simile a quello di **GMail**. (http://localhost:8888/html5-2013/html5/44-file-upload.html)

```
function upload(blobOrFile) {
  var xhr = new XMLHttpRequest();
  xhr.open('POST', '/server', true);
  xhr.onload = function(e) { ... };
  xhr.send(blobOrFile);
}

document.querySelector('input[type="file"]').addEventListener('change', function(e) {
  var blob = this.files[0];

  const BYTES_PER_CHUNK = 1024 * 1024;  // 1MB chunk sizes.
  const SIZE = blob.size;

  var start = 0;
  var end = BYTES_PER_CHUNK;

  while(start < SIZE) {
    upload(blob.slice(start, end));

    start = end;
    end = start + BYTES_PER_CHUNK;
  }
}, false);
})();
```

jQuery

jQuery è una **JavaScript Library**. [http://jquery.com/].

jQuery **semplifica** grandemente la programmazione Ajax ed è **easy to learn**.

```
$(document).ready(function(){
  $("p").click(function(){
    $(this).hide();
  });
});
```

Resources:
http://www.w3schools.com/jquery/jquery_examples.asp [w3c samples]
http://www.w3schools.com/jquery/jquery_ref_selectors.asp [selectors]

Demos:
http://www.bibeault.org/jqueryinaction/chapter2/lab.selectors.html [selectors]
http://www.bibeault.org/jqueryinaction/lab.move.and.copy.html [move DOM]
http://www.bibeault.org/jqueryinaction/chapter5/custom.effects.html [effects]
http://www.bibeault.org/jqueryinaction/chapter5/lab.effects.html [lab effects]
http://www.bibeault.org/jqueryinaction/chapter7/photomatic/photomatic.html [photogallery]
http://www.bibeault.org/jqueryinaction/chapter9/dimensions/lab.scroll.html [scroll]
http://www.bibeault.org/jqueryinaction/chapter9/ui/lab.draggables.html [draggables]
http://www.bibeault.org/jqueryinaction/chapter9/ui/lab.droppables.html [droppables]
http://localhost:8888/jquery_course_2014/