AJAX

Cos'è il Web 2.0

- **Web 2.0** = termine introdotto per la prima volta nel 2004 come titolo di una conferenza promossa dalla casa editrice O'Reilly
- L'idea è che ci si stia avviando verso una **nuova concezione** del web ("versione" 2.0), in contapposizione con la "vecchia" concezione ("versione" 1.0)
- Concetto **confuso e sfaccettato**… è difficile darne una definizione, generalmente si indicano semplicemente una serie di concetti emergenti
- NB: è gia… **online**! Google, Yahoo, eBay, Amazon, Napster, del.icio.us, Wikipedia, ecc. ecc.

Cos'è AJAX

- **AJAX** = "un pezzo dell'anima tecnologica del Web 2.0"…
- **AJAX** permette di dare, alle applicazioni web, l'interattività e la velocità delle applicazioni "desktop" (installate localmente sul vostro computer)
- **AJAX** = *Asynchronous JavaScript and XML*, coniato nel febbraio del 2005 da Jasse James Garrett, per descrivere un insieme di **applicazioni web dinamiche** basate sull'**interazione tra diverse tecnologie**:
  - **(X)HTML e CSS** per la visualizzazione della pagina
  - **DOM**, modificato attraverso **Javascript**, per offrire dinamicità alle pagina web
  - **XMLHttpRequest**, che consente al browser e al server di comunicare senza che la pagina venga ricaricata, permettendo la creazione di pagine web dinamiche più veloci

Cos'è AJAX

- **(X)HTML** = raccomandazione del W3C che può essere considerata un adattamento di HTML alle specifiche XML; prevede un uso più restrittivo dei tag HTML
- **DOM** = *Document Object Model* = standard ufficiale del W3C per la rappresentazione di documenti strutturati sul web (mi permette, per es, di manipolare gli elementi di una pagina web, quali le immagini, i campi dei moduli, ecc.)
- **XMLHttpRequest** = oggetto (insieme di API) che può essere usato da vari linguaggi di scripting dei browser (come Javascript) per scambiare dati (in formato testo o XML) con un web server, tramite protocollo HTTP

  $\Rightarrow$ **AJAX** è solo una **nuova etichetta** per riassumere l'utilizzo congiunto di **tecnologie preesistenti** e utilizzate ampiamente già da molto tempo

Cos'è AJAX

Applicazioni dinamiche tradizionali:

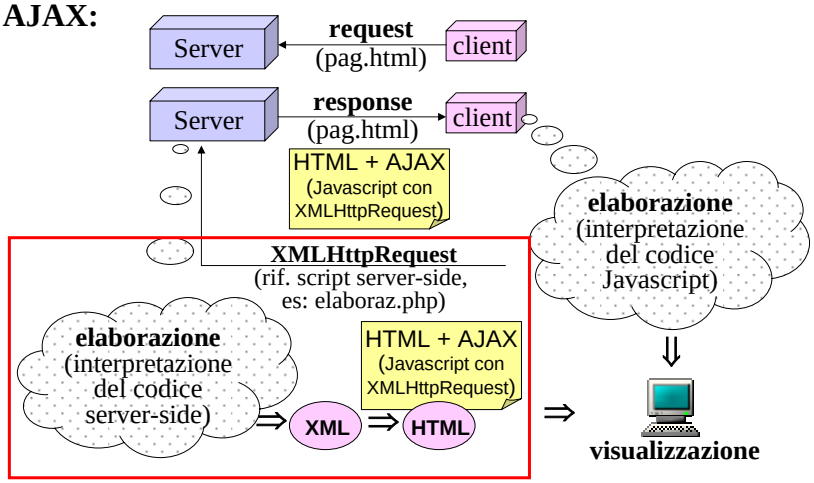
- per **ogni interazione** con l'utente (per es. click sul pulsante "Submit" di un modulo) inviano al server una **richiesta** per una nuova **pagina** (che conterrà la risposta del server)
- spesso la **risposta** del server rappresenta una **piccola parte** della nuova pagina, che però viene ricaricata per intero
- ciò comporta uno **spreco di banda** e un'**interfaccia utente** molto più **lenta** di quanto potrebbe essere

Applicazioni AJAX:

- sono in grado di inviare al web server **richieste asincrone** (mentre l'utente può continuare ad interagire con la pagina) e **parziali** (relative solo ai dati necessari)
- di conseguenza consentono un'**interazione più veloce** (la quantità di dati che è necessario inviare al/ricevere dal server è minore)

Cos'è AJAX

AJAX:



richiesta **asincrona** (non "blocca" l'interazione con l'utente)

AJAX: come funziona – primo esempio

l'utente digita città e via…

il sistema scrive automaticamente il CAP

AJAX: come funziona – primo esempio

In un'applicazione dinamica tradizionale (ASP, PHP, JSP):

- impossibile

In un'applicazione AJAX:

- quando l'utente digita la città e la via uno **script client-side** (Javascript) se ne accorge e invia (attraverso l'oggetto *XMLHttpRequest*) una **richiesta asincrona al server** nella quale si chiede il CAP associato a quella città/via
- l'utente può continuare ad interagire con la pagina
- quando il server ha trovato il CAP (probabilmente archiviato in un database) lo **invia allo script client-side** (sempre attraverso l'oggetto *XMLHttpRequest*) che lo inserisce nella pagina
- viene effettuata una **connessione asincrona con il server**; **la pagina non viene ricaricata** (rigenerata)

AJAX: XMLHttpRequest

Utilizziamo ora ***XMLHttpRequest*** per inviare una **richiesta asincrona** al web server (rif. file esAjax1.html):

1. Creiamo un oggetto *XMLHttpRequest* per **gestire la comunicazione (asincrona) con il web server**

   Purtroppo, la gestione dell'oggetto *XMLHttpRequest* **non** è uguale per i vari browser…

   In particolare, in Microsoft Internet Explorer l'oggetto *XMLHttpRequest* è restituito da un **ActiveXObject** [*] mentre negli altri browser (Mozilla, FireFox, Netscape, Opera, Safari, …) è supportato nativamente (come dovrebbe essere in MSIE7)

   [*] **ActiveX** è una tecnologia Microsoft per l'implementazione di particolari "controlli" (simili a plug-in) che offrono funzionalità specifiche; tali "controlli" possono essere incorporati ed utilizzati all'interno di varie applicazioni, tra cui Ms Internet Explorer

AJAX: XMLHttpRequest

⇒ per creare un oggetto *XMLHttpRequest* multi-browser:

```
function setXMLHttpRequest() {
  var xhr = null;
  // browser standard con supporto nativo
  if ( window.XMLHttpRequest ) {
    xhr = new XMLHttpRequest();
  }
  // MSIE con ActiveX
  else if ( window.ActiveXObject ) {
    xhr = new ActiveXObject("Microsoft.XMLHTTP");
  }
  return xhr;
}
var xhrObj = setXMLHttpRequest();
```

proprietà dell'oggetto *window*: restituisce un oggetto in grado di gestire una *XMLHttpRequest*

- con MSIE: *undefined*
- con Mozilla, …: *[XMLHttpRequest]*

proprietà dell'oggetto *window*: restituisce il controllo ActiveX in grado di gestire una *XMLHttpRequest*

- con MSIE: *function ActiveXObject() { [native code] }*
- con Mozilla, …: *undefined*

AJAX: XMLHttpRequest

2. Creiamo una **funzione** che viene invocata quando l'utente scrive la città e che **si connette al server**

```
<FORM ...>
 ... <INPUT TYPE="TEXT" ID="citta"
           onChange="callServerCity(this.value);">
 ... <INPUT TYPE="TEXT" ID="cap">
 ...
</FORM>

function callServerCity(c) {
  var url = "getCap.php?city="+c;
  ...
```

contenuto dell'attributo *value* del campo di testo *citta* (cioè la città scritta dall'utente!!!)

URL nella quale indichiamo la **pagina php** che contiene lo **script server side** che deve essere eseguito quando viene invocata la funzione *callserverCity(c)*
NB in questo caso passiamo gli passiamo anche un parametro che conterrà la città scritta dall'utente…

AJAX: XMLHttpRequest

```
...
xhrObj.open( "GET", url, true );
```

metodo della connessione

pagina server-side (php)

true = richiesta asincrona (lo script prosegue)

false = richiesta sincrona (lo script aspetta)

apriamo la connessione con il server

```
xhrObj.onreadystatechange = updatePage;
```

proprietà dell'oggetto *XMLHttpRequest* (mi serve per dire al server cosa fare quando ha finito di processare la richiesta)

indichiamo la funzione (*updatePage*) che il web server invocherà quando avrà finito la sua elaborazione

```
xhrObj.send( null );
```

il parametro (*city*) è già stato inviato nell'URL (qui, non avendo più parametri da inviare, inviamo *null*)

inviamo la richiesta al web server

```
}
```

AJAX: XMLHttpRequest

3. Gestiamo la risposta del web server = **definiamo la funzione che viene invocata dal server**

```
function updatePage() {
if (xhrObj.readyState == 4) {
    var risp = xhrObj.responseText;
    document.getElementById("cap").value = risp;
  }
}
```

3) scriviamo la risposta (il cap corrispondente alla città inviata nel parametro *city*) nel campo di testo corrispondente della form

2) leggiamo il valore della proprietà *responseText* dell'oggetto *xhrObj*, in cui il server mette la sua risposta

1) non facciamo nulla finché il server non ci dice che è pronto (*xhrObj.readyState == 4*)

AJAX: XMLHttpRequest

Ma cosa è successo sul server? Nella funzione *callServerCity*, quando abbiamo aperto la connessione (`xhrObj.open("GET", url, true);`) come secondo parametro gli abbiamo indicato una pagina php (`var url = "getCap.php?city="+c;`)… eccola (rif. file getCap.php):

```
<?
  $citta = $_GET["city"];
  $db = mysql_connect("localhost", "root", "") or
        die ("Non riesco a creare la connessione");
  mysql_select_db("caps") or die ("Non trovo il db");
  $sql = "SELECT * FROM codici WHERE citta='".$city."'";
  $ris = mysql_query($sql) or die ("Query fallita!");
  $riga = mysql_fetch_array($ris);
  echo $riga["cap"];
  mysql_close();
?>
```

leggiamo il parametro inviato in coda all'url

interroghiamo il DB

scriviamo sulla *response* il valore del campo "cap":
NB: in questo caso, la *response* è la proprietà *responseText* dell'oggetto *xhrObj*, in cui il server mette la sua risposta!

citta	cap
torino	10100
milano	20100
palermo	90100
napoli	80100
firenze	50100
bologna	40100

AJAX: note su Text e XML

La **comunicazione tra client e server** può riguardare:

- dati molto semplici → possiamo utilizzare semplici **stringhe**
- dati strutturati → possiamo utilizzare **oggetti XML**

Nel nostro esempio (rif. getCap.php) la **risposta del server** è molto semplice:

```
echo $riga["cap"];
```

- *echo* scrive nella proprietà *responseText* dell'oggetto *XMLHttpRequest* perché il *Content-Type* è definito (default) come *text/html*
- se vogliamo inviare, come risposta, un oggetto XML, dobbiamo definire il *Content-Type* a *text/xml*: in questo modo *echo* scrive nella proprietà *responseXML* dell'oggetto *XMLHttpRequest*

AJAX: note su Text e XML

Nel nostro esempio (rif. esAjax1.html) è la funzione *updatePage()* che **legge la risposta del server**:

```
var risp = xhrObj.responseText;
```

- in questo caso, il server ha scritto nella proprietà *responseText* dell'oggetto *XMLHttpRequest*
- se il server avesse scritto un oggetto XML nella proprietà *responseXML* dell'oggetto *XMLHttpRequest*, dovremmo leggere quella

```
var risp = xhrObj.responseXML;
```

e poi parsificare l'oggetto XML *risp* (con Javascript)

NB: `xhrObj.responseFormat` < `"text"` | `"XML"`

Memoria cache (e pagine web dinamiche)

Attenzione!

Ajax ha un pessimo rapporto con la **memoria cache del browser**…

NB: questo è vero di TUTTE le applicazioni dinamiche

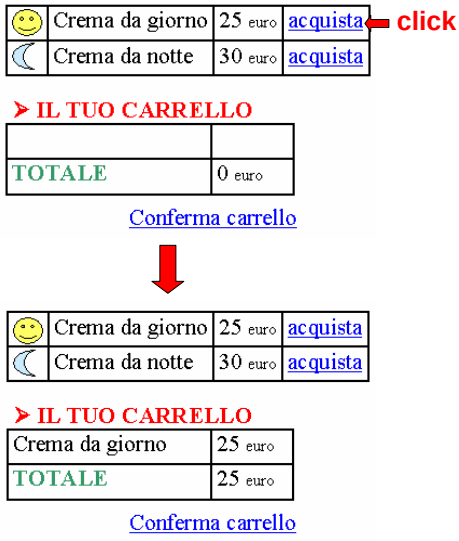
⇒ fate sempre attenzione all'effetto cache!!!

NB: la memoria cache si può:

- svuotare (menu del browser)
- disattivare (opzioni di configurazione del browser)
- impedire (forse…) che una pagina venga salvata nella cache → nella pagina AJAX (per es. in esAjax1.html):

```
<HEAD>
 ...
 <META HTTP-EQUIV="CACHE-CONTROL" CONTENT="NO-CACHE">
 ...
</HEAD>
```


AJAX: come funziona – secondo esempio



AJAX: come funziona – secondo esempio

In un'applicazione dinamica tradizionale (ASP, PHP, JSP):

- quando l'utente clicca su **acquista** parte una **richiesta al server**
- il server aggiorna il carrello, inserisce i nuovi dati nella pagina di risposta
- la (nuova) **pagina di risposta** viene inviata al client

In un'applicazione AJAX:

- quando l'utente clicca su **acquista** viene attivato uno **script client-side** (Javascript) che e invia (attraverso *XMLHttpRequest*) una **richiesta asincrona al server** nella quale si chiede di aggiornare i dati del carrello
- quando il server ha elaborato il nuovo carrello lo **invia allo script client-side** (sempre attraverso l'oggetto *XMLHttpRequest*) che lo inserisce nella pagina
- viene effettuata una **connessione asincrona con il server**; **la pagina non viene ricaricata** (rigenerata)