Tecnologie Cloud e Mobile

Lez. 06

AJAX e JSON

Giuseppe Psaila

Università di Bergamo

giuseppe.psaila@unibg.it

Siti Web Statici

- Insieme di pagine HTML, il cui contenuto è staticamente fissato dal Web Master
- Utili per realizzare i “Siti Vetrina”
- Inadatti per realizzare Servizi per l’utente

Siti Web Dinamici

- Insieme di pagine HTML, il cui contenuto è generato dinamicamente da programmi che operano sul Web Server
- La pagina viene interamente rigenerata ad ogni richiesta di elaborazione
- Hanno consentito la realizzazione delle prime “Applicazioni Web”

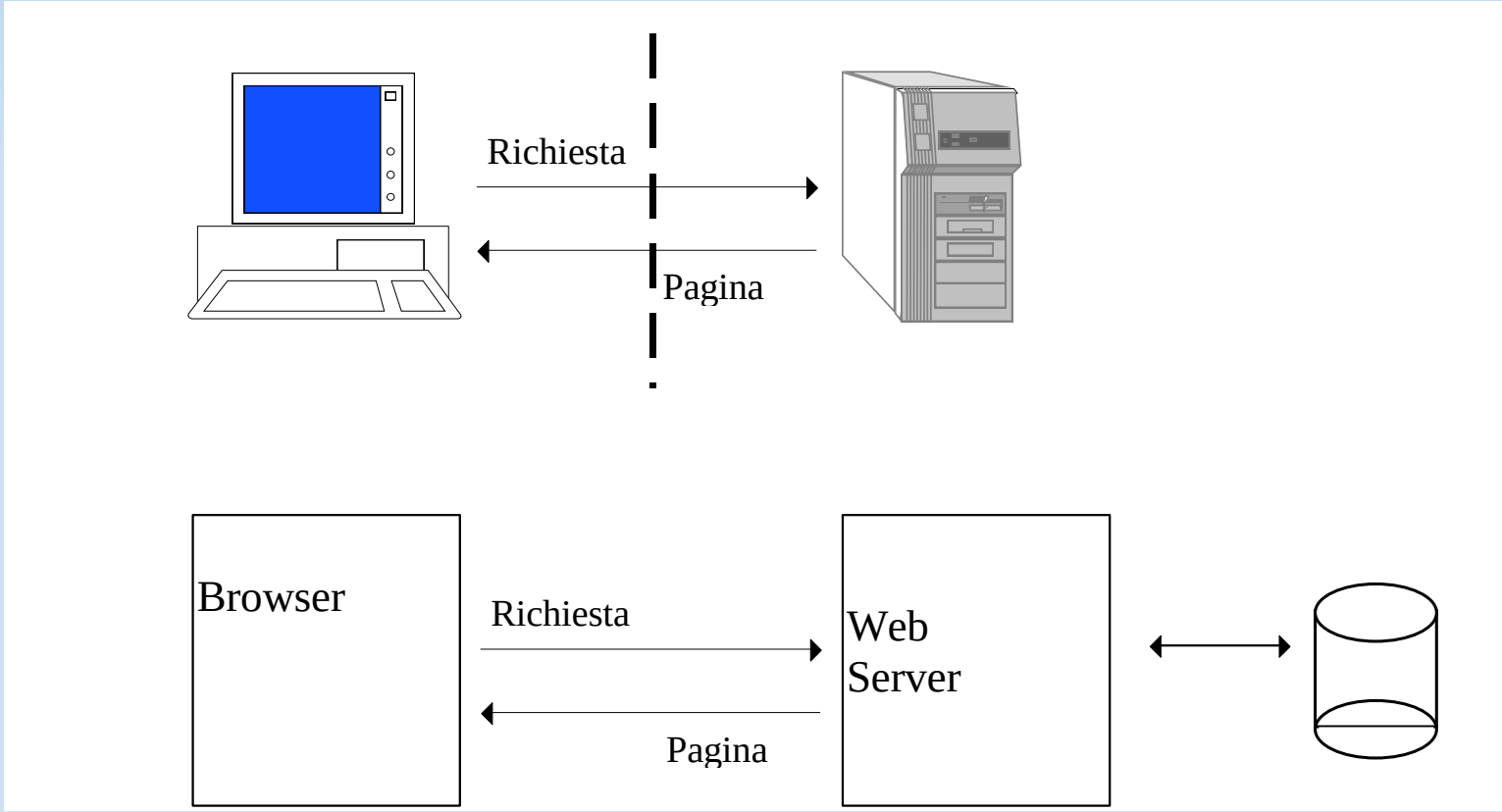
Architetture

Le architetture per realizzare siti web sono evolute di pari passo: attraverso le tre proposte seguenti

- 1-tier Architecture
- 2-tier Architecture
- 3-tier Architecture

1-tier Architecture

- La più semplice
- Adatta per i siti statici



Che cosa è un tier?

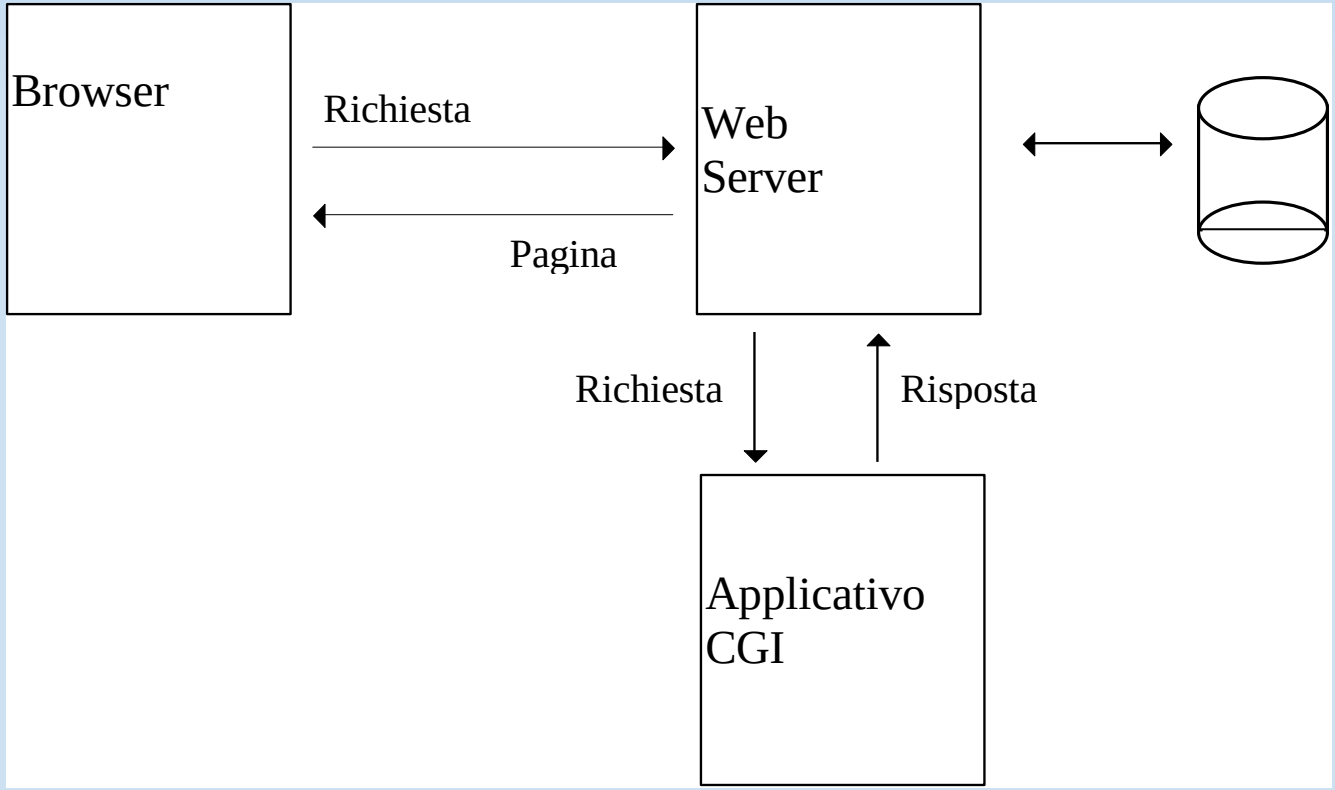
- Traduzione di tier: «annodatore»
- In pratica, la connessione tra due strati software
- Nella 1-tier architecture, il tier è il protocollo HTTP

1-tier Architecture

- Il browser invia una richiesta HTTP
- Con la quale richiede una risorsa (pagina, immagine, altro,…)
- Il server riceve la richiesta
- Se la risorsa è presente sul disco,
- La invia al client, impacchettandola nella Risposta HTTP

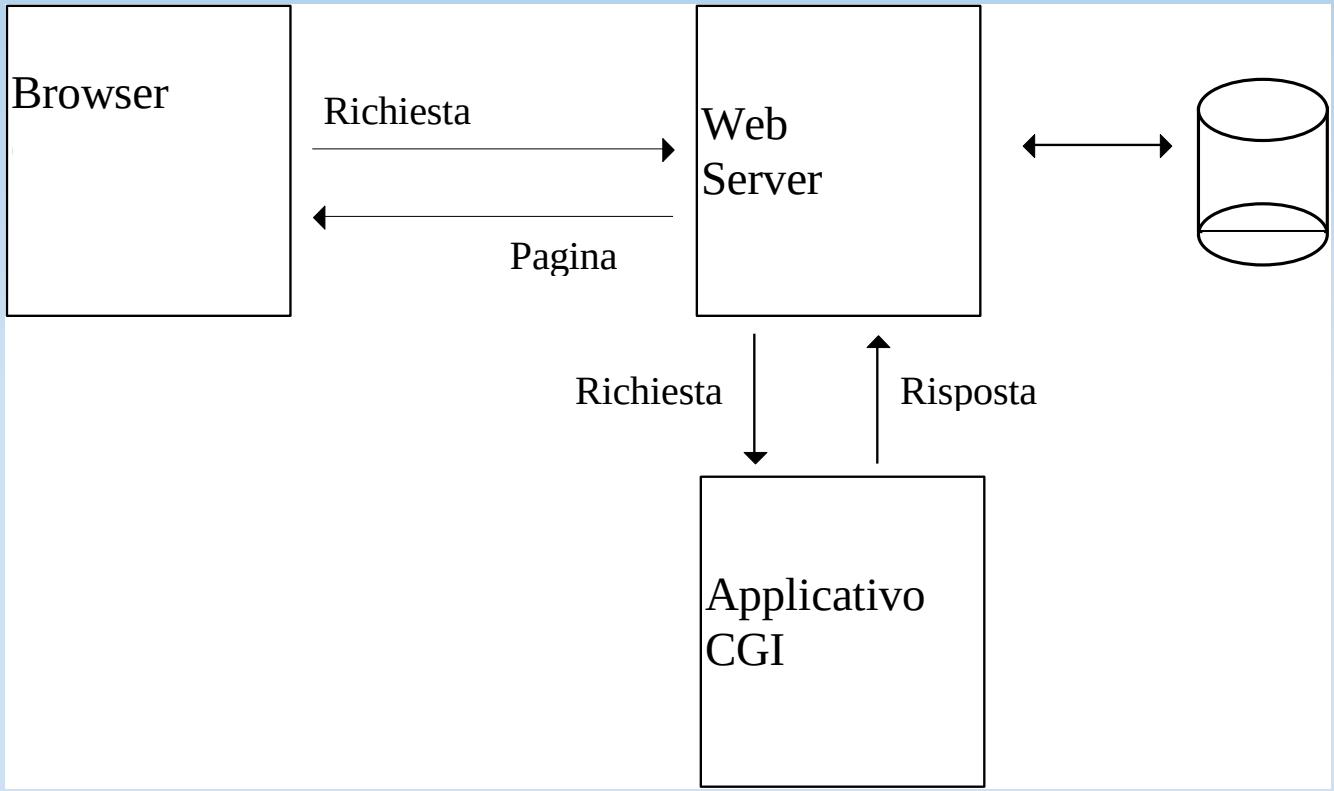
2-tier Architecture

- Adatta per i siti dinamici
- Dove i contenuti devono essere generati da programmi che lavorano sul server



2-tier Architecture

- Il primo tier è il protocollo HTTP
- Il secondo tier è il protocollo CGI (Common Gateway Interface) tra il web server e l'applicativo CGI



2-tier Architecture

- Il browser invia una richiesta HTTP al server
- Il server riceve la richiesta HTTP, vede che è per un programma
- Attiva il programma (Applicativo CGI) e gli passa il contenuto della richiesta

2-tier Architecture

- L’applicativo CGI genera la pagina HTML (o altro contenuto)
- Il server impacchetta l’output dell’applicativo CGI nella Risposta HTTP
- La Risposta HTTP viene inviata al browser sul client

La Richiesta HTTP

- I dati delle form in HTML vengono inviati con due metodi distinti, pur usando sempre la Richiesta HTTP
- Metodo GET
- Metodo POST

Esempio di Form (methodo GET)

```
<form method="GET" action="…"
  onSubmit="return controlla();">
 <input type="text" name="email"/>
 <input type="text" name="nome"/>
 <input type="submit" value="Invia"/>
</form>
```

Richiesta

- Con il metodo GET, il corpo della richiesta è vuoto
- Nello header, il campo URL contiene l'indirizzo della risorsa con l'aggiunta dei campi della form
- Dopo il ?, triplette ***campo=valore*** separate da &

http://...?**email=a@unibg.it&nome=Pippo**

Il Metodo GET

- Per fare un parallelo «cartaceo», è come se
- Prepariamo una busta;
- Sul fronte mettiamo tutti i dati, il destinatario e altre informazioni accessorie
- Ma la busta la lasciamo vuota
- Infatti, il destinatario prende le informazioni dalla busta stessa

Metodo POST

- Se non si vuole che i dati siano visibili nella barra degli indirizzi
- Oppure perché il valore può essere troppo grosso (per esempio aree di testo oppure file)
- Si ricorre al metodo POST

Richiesta HTTP

- Il corpo della richiesta non è più vuoto, ma contiene i campi della form **email=a@unibg.it&nome=Pippo**
- L’URL del destinatario non viene modificato

Protocollo CGI

- Il protocollo CGI è stato il primo protocollo per far comunicare il web server con le applicazioni lato server
- All’epoca (1995) erano scritte in C
- Il programma C comunicava attraverso standard input e standard output

Protocollo CGI

- Metodo GET: i valori dei campi erano ricevuti come variabili d’ambiente
- Metodo POST: il contenuto del corpo della richiesta era ricevuto come standard input
- L’output del programma era inviato sullo standard output

Ora che si fa?

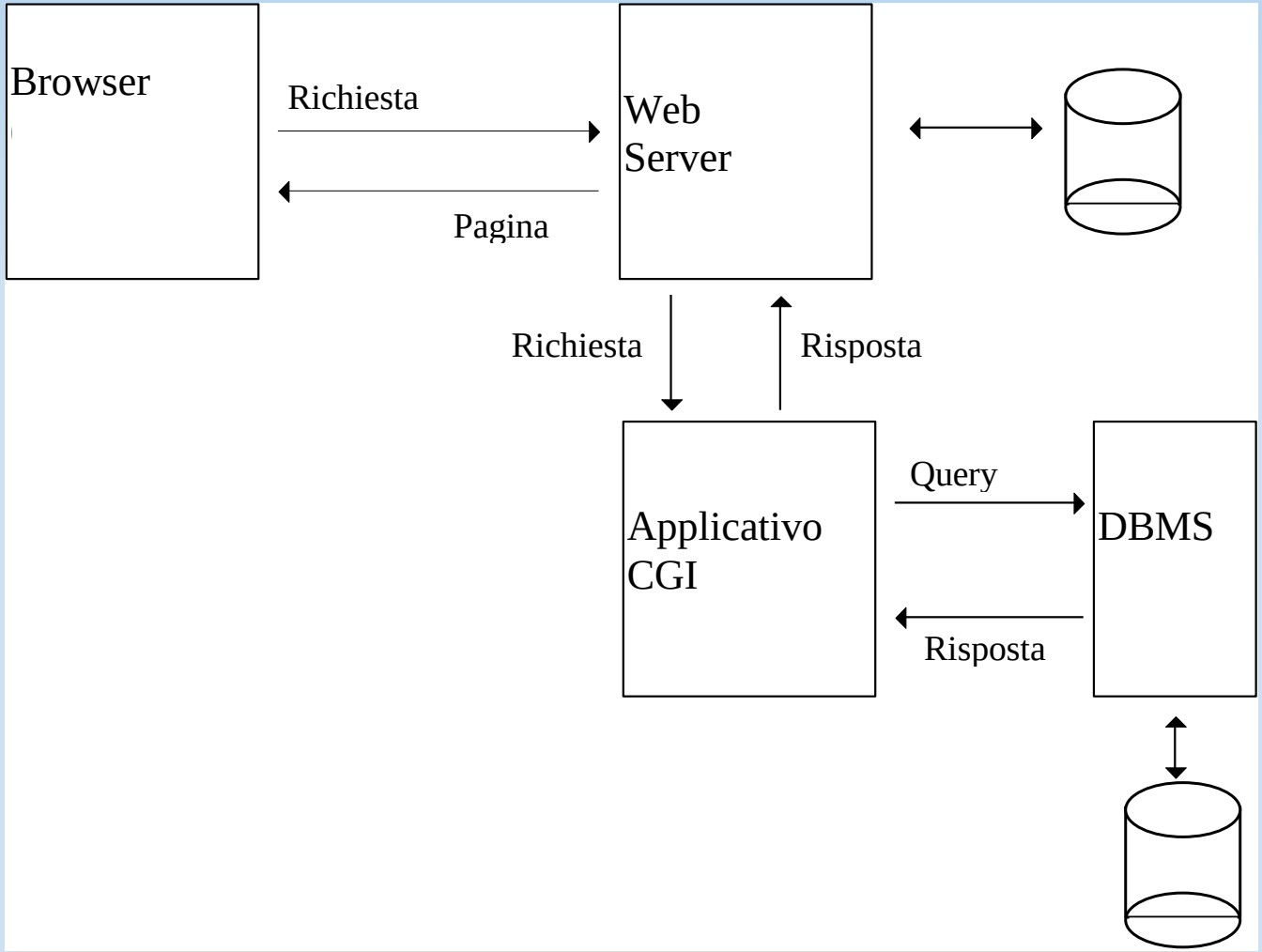
- Negli anni sono stati sviluppati i «server-side script»
- Idea: annegare il codice procedurale in uno scheletro HTML
  - Esempi:
  - PHP
  - JSP (basate su Java), tradotte nelle Servlet Java
  - VBA o C# (Microsoft)

Ora che si fa?

- Tuttavia, per capire il ruolo,
- Continuiamo a parle di Applicativo CGI,
- Anche se il protocollo CGI, di fatto, nonn lo usiamo più (ma i web server lo supportano ancora)

3-tier Architecture

- Interazione con il DBMS
- Da parte dello Applicativo CGI



3-tier Architecture

- Il terzo tier è la connessione con il DB
- La connessione con il DB avviene in vari modi: il protocollo più generico si chiama ODBC
  Open DataBase Connectivity

3-tier Architecture

- Con Java, il protocollo di base si chiama JDBC
- Poi si usa il bridge specifico
  - JDBC:ODBC
  - JDBC:Postgresql

3-tier Architecture

- Il browser invia una richiesta HTTP al server
- Il server riceve la richiesta HTTP, vede che è per un programma
- Attiva il programma (Applicativo CGI) e gli passa il contenuto della richiesta

3-tier Architecture

- L’applicativo CGI apre la connessione con il DB
- Invia le query e riceve le tabelle
- Genera l’output e lo invia al web server
- Che a sua volta lo impacchetta nella Risposta HTTP e lo invia al browser

AJAX

Prime Applicazioni Dinamiche

- Le prime applicazioni web dinamiche erano un po’ scomode da usare
- Tutto era fatto dal server, quindi ogni azione fatta dall’utente richiedeva di aspettare l’arrivo della pagina rigenerata

Ora

- Il codice JavaScript associato alla pagina
- comunica direttamente con il server
- e aggiorna di conseguenza le informazioni nella pagina
- in modo trasparente all’utente

L’Oggetto XMLHttpRequest

- Per consentire al codice JacaScript di comunicare con il server
- è stato aggiunto un oggetto specifico all’interprete JavaScript:
- l’oggetto XmlHttpRequest

AJAX

- Asynchronous
  JavaScript
  And
  XML

Asynchronous

- Il client invia una richiesta al server, ma la risposta arriva in modo asincrono
- In attesa della risposta, il client può continuare a lavorare

JavaScript

- Pensato per rendere a tutti gli effetti i codici JavaScript delle applicazioni client-server
- È stato necessario estendere l’oggetto predefinito Window con specifiche funzionalità per gestire la comunicazione asincrona

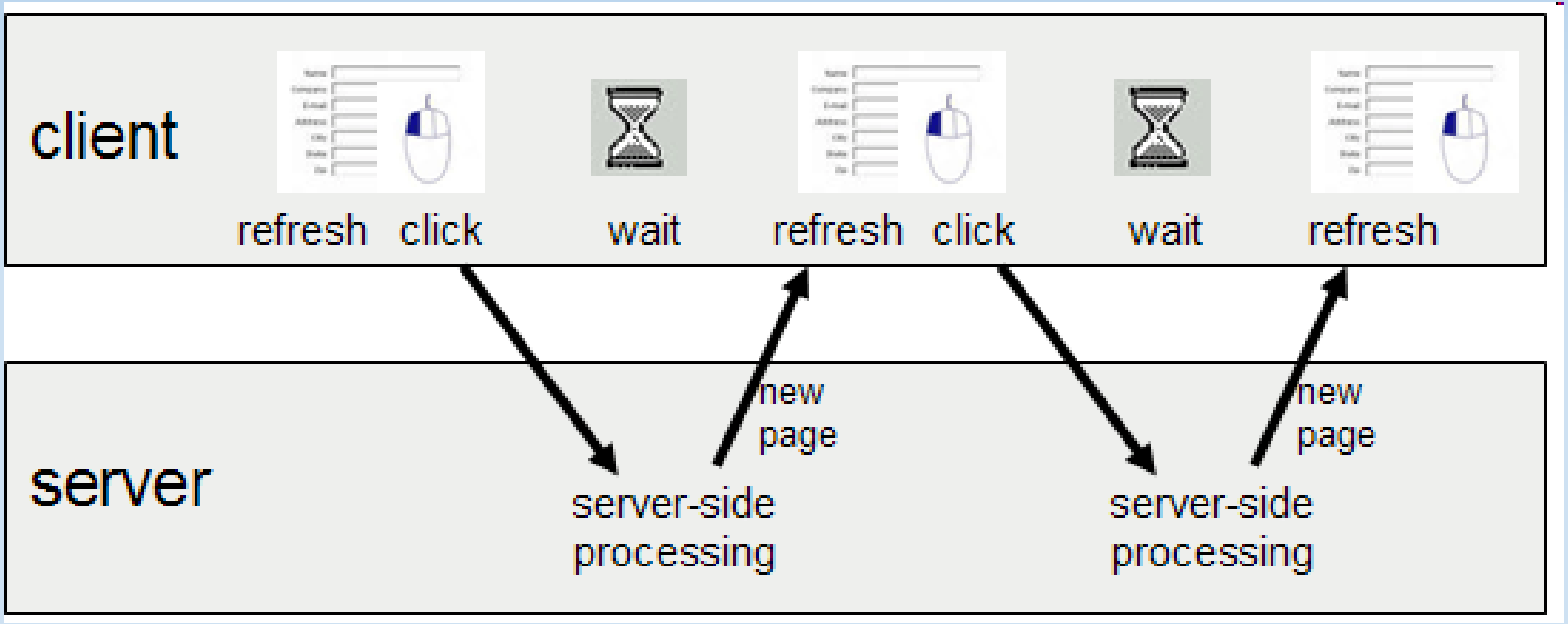
XML

- I messaggi inviati dal server sono in XML,
- Non sono infatti pagine web
- Ma i contenuti che il server manda in risposta alla richiesta del client
- Il client JavaScript ha il compito di usarli/mostrarli nella pagina HTML

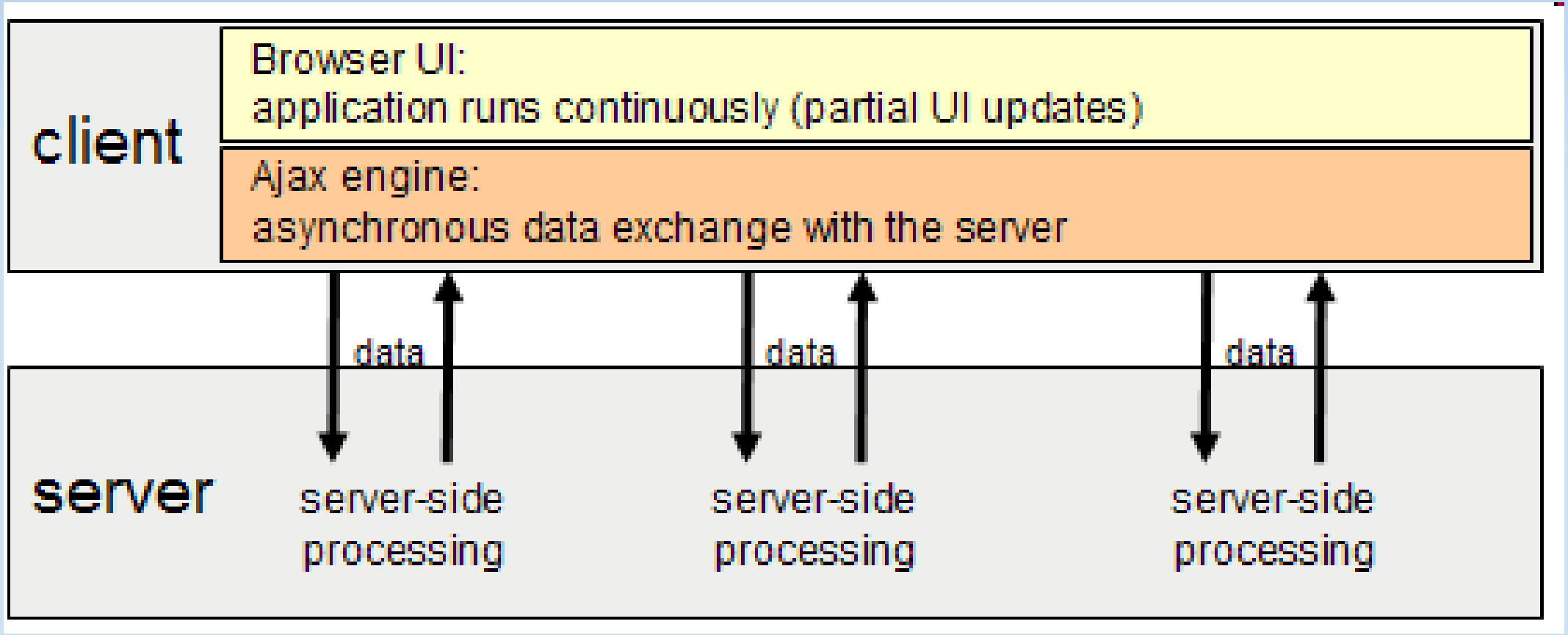
XML

- I documenti XML ricevuti vengono rappresentati con la struttura dati DOM (Document Object Model)
- Questo ha richiesto una ulteriore estensione dell'interprete JavaScript
- Usando il metodo POST, i documenti XML possono essere mandati dal client al server nella richiesta HTTP.

HTML: Click, Wait, Refresh



Ajax-powered User Experience



Tecnologia AJAX Base

L’Oggetto XmlHttpRequest

- Gli interpreti JavaScript dei browser sono stati estesi per gestire la comunicazione con il server,
  fare il parsing del documento XML ricevuto dal server e costruire la struttura dati DOM
- L’oggetto built-in dell’interprete chiamato

  XmlHttpRequest

  fa questo lavoro (come campo di Window)

L’Oggetto XmlHttpRequest

Uso:

- Si clona l’oggetto con l’istruzione

  new XmlHttpRequest()

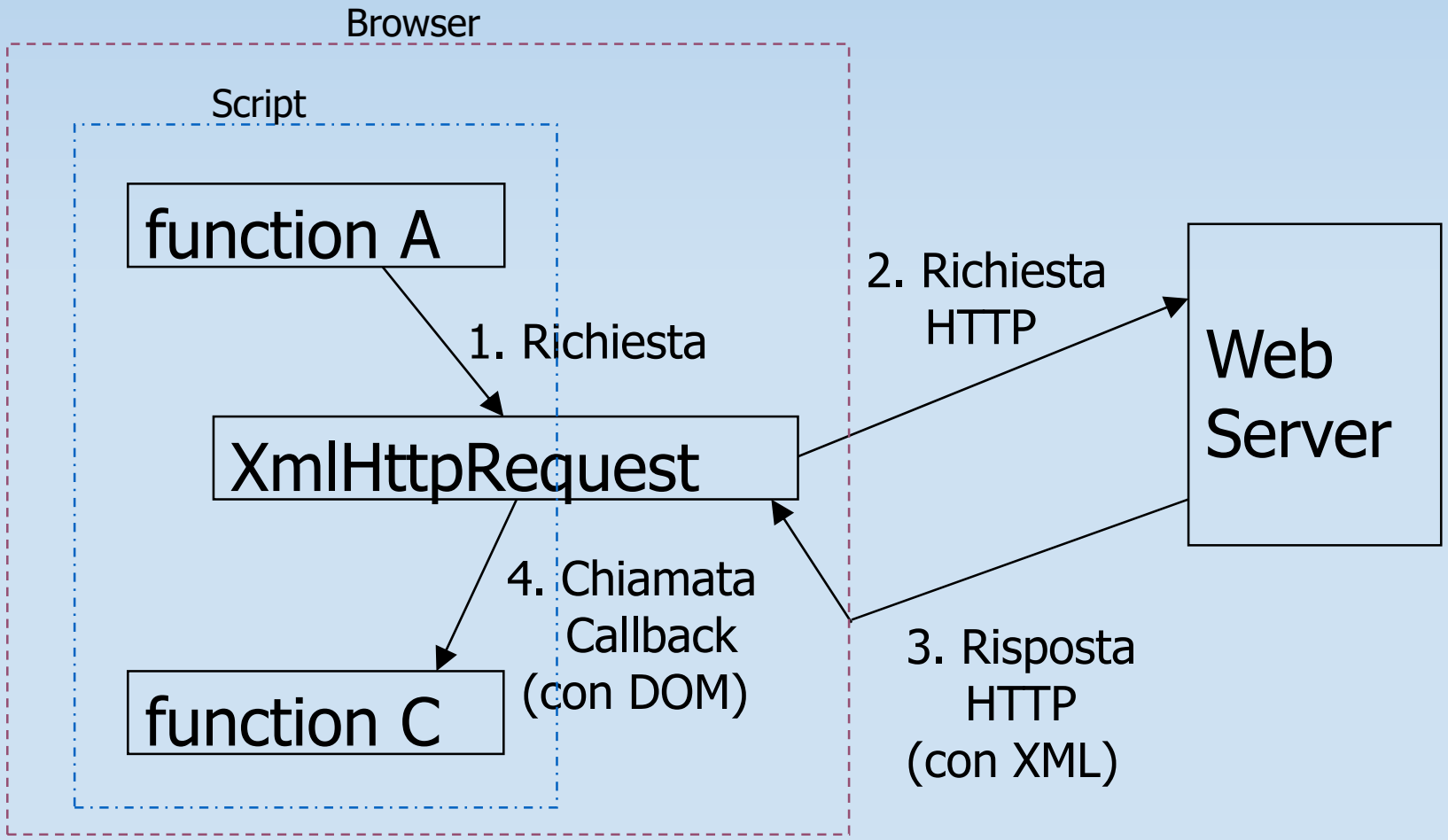
- Si impostano le proprietà e i metodi
- Si chiama il metodo **open**, per aprire la connessione
- Si chiama il metodo **send** per inviare la richiesta.

L’Oggetto XmlHttpRequest

Schema della comunicazione:

- Lo script invia una richiesta al server tramite l’oggetto XmlHttpRequest
- L’oggetto XMlHttpRequest invia la richiesta e aspetta la risposta
- Quando la risposta arriva, chiama una funzione dello script detta «callback», fornendo il messaggio ricevuto

L'Oggetto XmlHttpRequest



L’Oggetto XmlHttpRequest

Tipo di comunicazione:

- **Sincrona**: l’oggetto XmlHttpRequest invia la richiesta HTTP e aspetta la risposta (lo script è bloccato) - **DEPRECATA**
- **Asincrona**: l’oggetto XmlHttpRequest passa su un thread parallelo; l’interprete JavaScript non si blocca e lo script continua a lavorare

L’Oggetto XmlHttpRequest

Gestione della risposta (messaggio XML):

- L’oggetto XmlHttpRequest riceve la risposta HTTP e analizza il contenuto
- Se questo è un documento XML, costruisce l’albero DOM (campo **`responseXML`**)
- In ogni caso lo tratta come testo (campo **`responseText`**)

L’Oggetto XmlHttpRequest

- Stato della comunicazione:
  campo **`readyState`**
- Se il valore è 4, indica che la risposta è arrivata
- Quando il suo valore cambia, l’oggetto XmlHttpRequest chiama il metodo
  `onreadystatechange`
  cioè la funzione di callback interna

L’Oggetto XmlHttpRequest

- Valori di **`readyState`**

0 (Uninitialized)

Oggetto non inizializzato (metodo **`open`** non invocato)

1 (Open)

Metodo **`open`** invocato, ma metodo **`send`** non invocato

L’Oggetto XmlHttpRequest

- Valori di **`readyState`**

2 (Sent)

Metodo **`send`** invocato, ma i campi **`responseText`** e **`responseXML`** non hanno valore

3 (Receiving)

Alcuni dati ricevuti, ma i campi **`responseText`** e **`responseXML`** non hanno valore

L’Oggetto XmlHttpRequest

- Valori di **`readyState`**

4 (Loaded)

Dati ricevuti, i campi **`responseText`** e **`responseXML`** (se il documento ricevuto è un XML ben formato) hanno valore

L’Oggetto XmlHttpRequest

- Campo **`onreadystatechange`**:
  la funzione da chiamare quando **`readyState`** cambia
- Campo **`status`**:
  il codice HTTP (200 = OK)
- Campo **`statusText`**:
  il testo associato al codice HTTP (per 404, “Not Found”)

L’Oggetto XmlHttpRequest

Metodi Principali:

- **`open(requestMethod, url, asynchronousFlag, username, password)`**

  Apre la connessione

L’Oggetto XmlHttpRequest

Metodi Principali:

- **`send(bodyContent)`**
  Invia la richiesta; **`bodyContent`** è il contenuto della richiesta (**`null`** se non presente)

L’Oggetto XmlHttpRequest

Metodi Principali:

- **`abort()`**
  Interrompe la sequenza di comunicazione; **`readyState`** viene reimpostato a 0

L’Oggetto XmlHttpRequest

Altri Metodi:

- **`getAllResponseHeaders()`**
  Restituisce una stringa con tutte le intestazioni (headers) della risposta HTTP, separati da new line

L’Oggetto XmlHttpRequest

Altri Metodi:

- **`getResponseHeader(headerField)`**
  Restituisce una stringa con l’intestazione (header) della risposta HTTP specificata come parametro

L’Oggetto XmlHttpRequest

Altri Metodi:

- **`setRequestHeader(headerField, headerValue)`**

  Imposta una intestazione (header) della richiesta HTTP, specificando nome e valore

L’Oggetto XmlHttpRequest

Altri Metodi:

- **`overrideMimeType(mimeType)`** sostituisce il mimeType del contenuto della richiesta

La Funzione makeAjaxRequest

- Tipicamente, non si lavora direttamente al livello basso dell'oggetto XmlHttpRequest
- Conviene alzare un po' il livello
- Di seguito, vediamo una funzione molto comune, che si trova facilmente in rete, per impostare le chiamate AJAX
- E' utile comprenderla, ma NON usatela professionalmente

makeAjaxRequest

La funzione riceve i seguenti parametri

- **`url`**, l'URL cui inviare la richiesta
- **`callback_function`**, la stringa con il nome della funzione di callback
- **`return_xml`**, un booleano che indica se ricevere (true) un documento XML
- **`xml_msg`**, il testo con l'eventuale messaggio XML da inviare

makeAjaxRequest

La funzione riceve i seguenti parametri

- **`method`**, il metodo HTTP usato ('GET' o 'POST')

La Funzione di Callback

La funzione di callback riceve due parametri:

- **`msg`** con il messaggio ricevuto (se testuale) o l'albero DOM (se XML)
- **`type`** il tipo del messaggio: 'text' o 'xml'

makeAjaxRequest

```
function makeAjaxRequest(url,
 callback_function, return_xml,
   xml_msg, method)
{ var http_request = false;

   if (window.XMLHttpRequest){
      http_request =
          new XMLHttpRequest();
```

makeAjaxRequest

```
 if (
   http_request.overrideMimeType)
 { http_request.
    overrideMimeType('text/xml');
 }
} else
```

makeAjaxRequest

```
if (window.ActiveXObject) { // IE
    try {
      http_request = new
 ActiveXObject("Msxml2.XMLHTTP"); }
    catch (e) {
       try { http_request = new
 ActiveXObject("Microsoft.XMLHTTP");}
         catch (e) {} } }
```

makeAjaxRequest

```
if (!http_request) {
  alert('Unfortunatelly your ' +
'browser doesn\'t support ' +
 'this feature.');
   return false;
 }
```

makeAjaxRequest

```
http_request.onreadystatechange =
   function()
  { if (http_request.readyState == 4)
    { if (http_request.status == 200)
      { if (return_xml) {
        eval(callback_function +
'(http_request.responseXML, "xml")');
} else
```

makeAjaxRequest

```
 { // rispsota testuale
eval(callback_function +
 '(http_request.responseText,
 "text")');
 } } else
{ alert('There was a problem with ' +
 'the request.(Code: ' +
 http_request.status + ')` );
} } }
```

makeAjaxRequest

```
 if(xml_msg != null)
      method = 'POST';
  http_request.open(method,
     url, true);
  // se xml_msg nullo, non invia
  // niente
  http_request.send(xml_msg);
}
```

Non Usate makeAjaxRequest

La funzione makeAjaxRequest non è da usare, perché:

- Non è robusta
- Non fa le chiamate cross-domain
- Non gestisce in automatico la correttezza dei contenuti ricevuti (rischio di JavaScirpt Injection)

Quindi, non fatela così ma con JQuery

JSON

AJAX Senza XML

- Gestire un documento XML, per quanto già nella rappresentazione DOM, è complicato
- Sia lato client che lato server
- Pertanto, è accaduto che al posto di XML i programmatori abbiano iniziato a usare un formato diverso
- JSON

JSON

- JavaScript
- Object
- Notation

JSON

- Deriva dalla terza forma per creare oggetti in JavaScript
- Ma non consente di specificare metodi
- Un documento JSON contiene solo dati
- Perché JSON?
- Perché con poco codice si converte un documento JSON in un oggetto JavaScript

Esempio di Oggetto

```
{"bindings":
   [ {"ircEvent": "PRIVMSG",
      "method": "newURI",
      "regex": "http://.*"},
     {"ircEvent": "PRIVMSG",
      "method": "deleteURI",
      "regex": "delete.*"} ] }
```

Formato JSON

- Nomi dei campi tra virgolette (no apici singoli)
- Campi semplici: numeri, stringhe (sia con doppi apici che apici singoli)
- Camapi complessi: oggetti annidati
- Campi array: array di valori semplici e/o oggetti

Assegnare un Oggetto

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

Stringa JSON

- Volendo, possiamo creare una stringa che contiene un documento JSON
- Come se la stringa arrivasse dal server
- E’ una versione «serializzata» dell’oggetto

Stringa JSON

```
var myJSONText =
'{"bindings":[ {"ircEvent":
 "PRIVMSG", "method": "newURI",
 "regex": "http://.*"},
 {"ircEvent": "PRIVMSG",
 "method": "deleteURI","regex":
 "delete.*"} ] }';
```

Deserializzare la stringa

- Per convertire la stringa in oggetto, si può usare la funzione **`eval`** nativa di JavaScript, che esegue il codice nella stringa di testo

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

Deserializzare la stringa

- Ma la funzione **`eval`** è problematica in termini di sicurezza.
- Conviene usare una libreria, chiamata JSONparser, che fornisce due funzioni molto utili:
- Object JSON.parse(JSONtext) da testo JSON a oggetto
- String JSON.stringify(object) da oggetto a stringa

Deserializzare/Serializzare

- Per convertire la stringa in oggetto

```
var myObject =
   JSON.parse( myJSONtext );
```

- Per ottenere la stringa da un oggetto

```
var myText =
   JSON.stringify( myJSONObject );
```