SEMINARIO INTRODUZIONE AD AJAX

Linguaggi per il Web

Ingegneria Informatica, Ingegneria dell'Informazione,
Sapienza Università di Roma, sede di Latina

17 Maggio 2024

Corrado Di Benedetto

ARGOMENTI

- Introduzione
- XMLHttpRequest
- JSON
- Framework Ajax

INTRODUZIONE

- **AJAX** è l'acronimo di ***A****synchronous* ***J****avaScript* ***A****nd* ***X****ML*

- E' stato introdotto nel 2005 da *Jesse James Garrett* (padre di AJAX) e reso popolare, sempre nello stesso anno, da Google

- Permette di creare una interfaccia web dinamica, ad **esempio** l'autocompletamento della *search box* di Google con una lista di suggerimenti del server

CHE COSA È AJAX ?

Non è una singola tecnologia, ma bensì una **collezione di tecnologie**:

1. **JavaScript:** per interagire con il browser e gestire gli eventi
2. **XHTML + CSS:** per la presentazione della pagina web
3. **DOM:** per accedere e manipolare la struttura XHTML della pagina web
4. **XML:** il formato per scambiare i dati tra il server e il client
5. **Oggetto XMLHttpRequest:** per scambiare in modalità *asincrona* i dati tra il server e il client

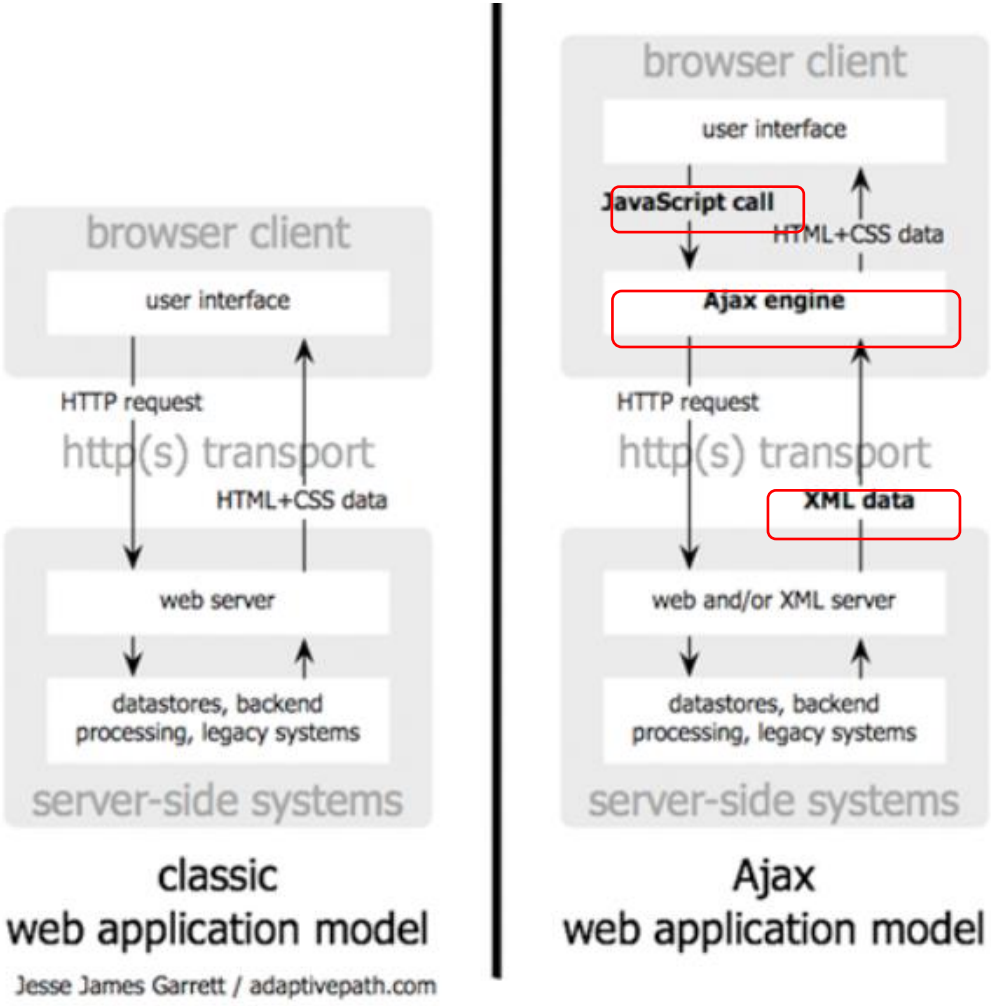
CHE COSA È AJAX ?

- Oggi ha un significato estensivo che comprende tutte le tecnologie native dei browser che permettono una **comunicazione asincrona** con un server

- E' **asincrono,** ovvero i dati scambiati fra client e server sono caricati in background senza bloccare il comportamento della pagina

- A differenza dell'approccio classico permette l'**aggiornamento dinamico** di una pagina web senza il completo ricaricamento

CHE COSA È AJAX ?

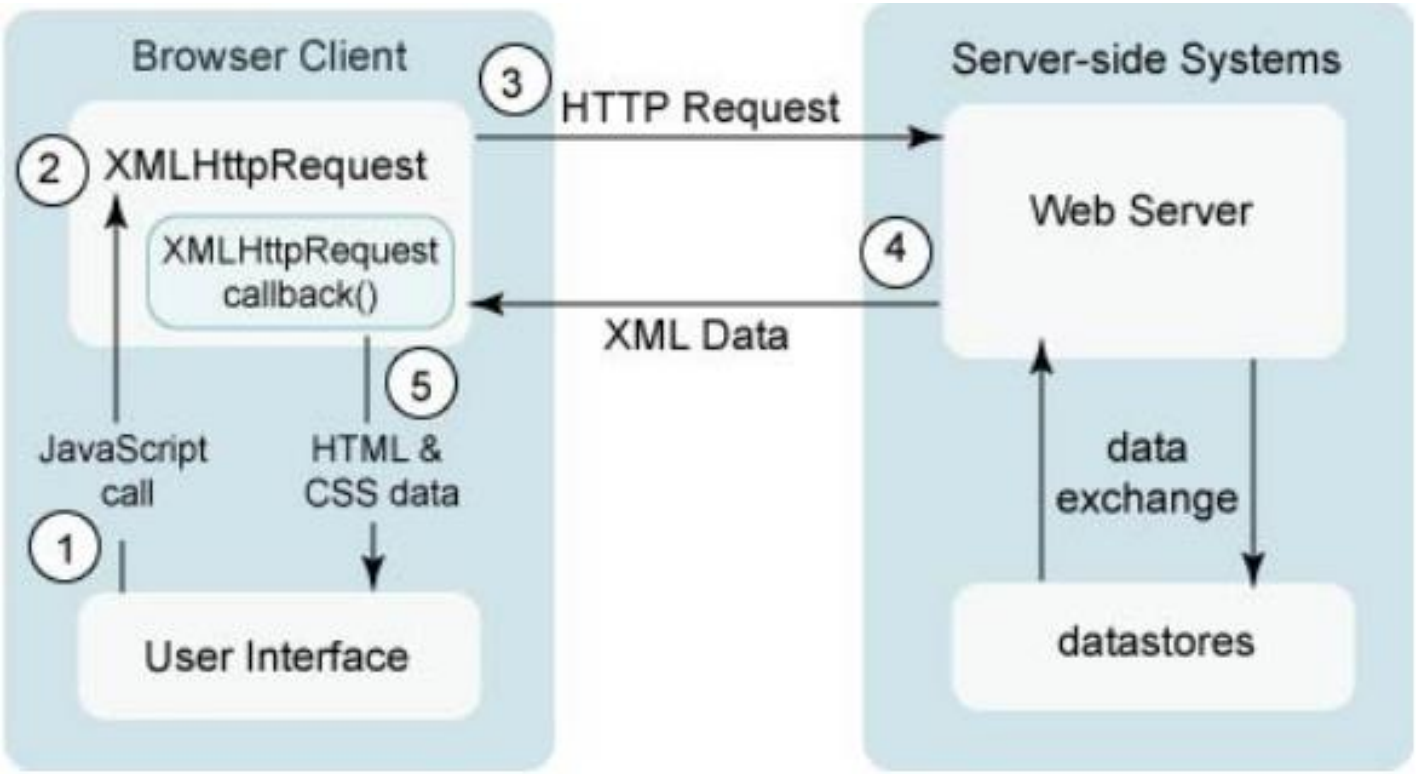


CHE COSA È AJAX ?

- Le tecnologie di base di Ajax sono JavaScript, XML e XHTML, ma l'uso di JavaScript e del XML non è obbligatorio

- Ajax permette lo scambio di dati con il server usando anche altre tecnologie, come ad esempio **JSON** (JavaScript Object Notation)

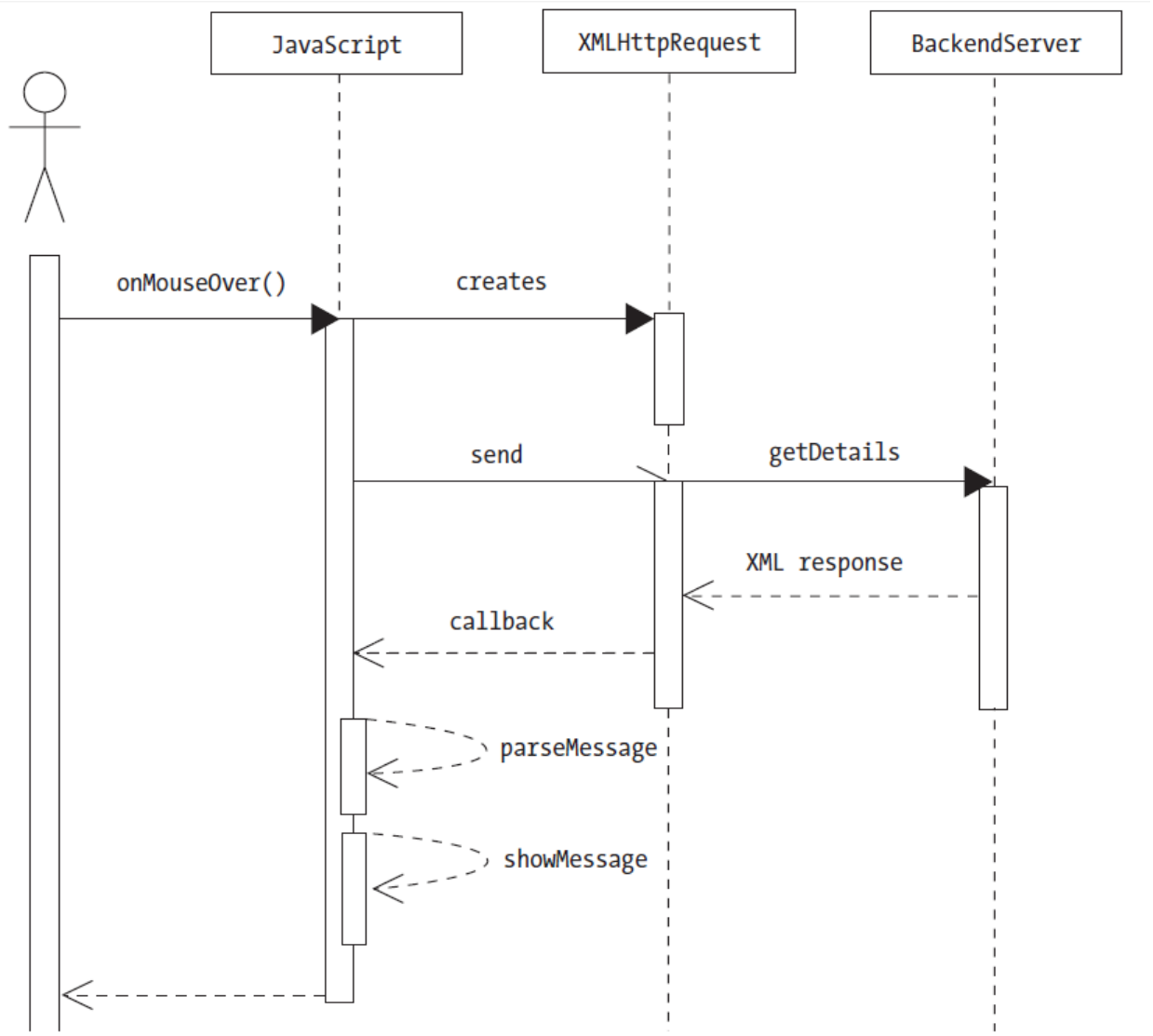
COME FUNZIONA AJAX ?



COME FUNZIONA AJAX ?

1. L'***utente*** genera un evento, ad esempio il passaggio del mouse su un elemento HTML, a cui corrisponde un gestore di eventi ***JavaScript***
2. Il gestore di eventi crea un oggetto ***XMLHttpRequest***
3. Esso interagisce attraverso una *richiesta asincrona* con il server di ***BackEnd***
4. Un componente software di ***BackEnd*** (es. PHP, ASPX, JSP) riceve e elabora la richesta e poi risponde in XML
5. L'oggetto ***XMLHttpRequest*** riceve ed elabora l'XML di risposta e poi aggiorna il DOM

COME FUNZIONA AJAX ?



DOVE USARE AJAX ?

- **Form**: Aumenta sensibilmente le prestazioni di una form HTML
- **Comunicazione**: Utile nella progettazione di componenti software per la comunicazione, ad esempio chat, bottoni di voto, messaggi thread, rating, ecc.
- **News**: RSS feeds può essere gestito con tecnologia Ajax (es. Google News)
- **Manipolazione dei dati**: Ad esempio l'ordinamento o il filtraggio dei dati di una tabella oppure l'autocompletamento di un campo di una form HTML

DOVE USARE AJAX ?

Osservazioni

- Ajax non è la soluzione ad ogni problema
- La gestione di grandi quantità di dati con Ajax può portare a problemi di prestazioni o di altro tipo
- Usare Ajax solo quando i widget JavaScript tradizionali non sono sufficienti o quando bisogna gestire i dati scambiati con il server

ESEMPI DI TECNOLOGIA AJAX

- Applicazioni web che implementano Ajax:

Google

- Gmail e Mappe, Calendario, Home Page Personalizzate e Search Box Google

Yahoo

- Home Page di Yahoo, un gran numero di personalizzazioni e di caratteristiche, come le anteprime delle e-mail

Altri Esempi

- Youtube, Facebook, …

PREGI

Usabilità

- Interattività

Velocità

- Minore quantità di dati scambiati
- Parte della computazione sul client

Portabilità

- Supportato dai principali browser
- Indipendente dalla piattaforma
- Non richiede plug-in

DIFETTI

Usabilità

- Non funziona il pulsante *back* e i segnalibri
- I motori di ricerca non indicizzano i contenuti dinamici

Accessibilità

- Non supportato dai browser non-visuali
- Richiede meccanismi di accesso alternativi

Configurazione

- JavaScript abilitato
- Oggetti ActiveX abilitati in IE

Compatibilità

- Test sui diversi browser
- Richiede funzionalità alternative per i browser che non supportano JavaScript

COMPATIBILITÀ DEI BROWSER

Compatibili con Ajax:

- *Internet Explorer* dalla versione 5
- *Mozilla Firefox* dalla versione 7.1
- *Konqueror* dalla versione 3.2
- *Safari* dalla versione 1.2
- *Opera* dalla versione 8.0
- *Chrome*

Non compatibile con Ajax:

- Versioni precedenti a quelle viste sopra
- Tutti i browser testuali
- Tutti i browser per disabili visivi (screen-reader, browser vocali, …)
- Tutti i browser precedenti al 1997

ARGOMENTI

- Introduzione
- XMLHttpRequest
- JSON
- Framework Ajax

OGGETTO XMLHTTPREQUEST

- E' di fondamentale importanza in Ajax, tutti i browser moderni lo supportano nativamente (IE5 e IE6 usano un ActiveX)

- E' usato per scambiare i dati in *background* con un server o aggiornare parti di pagina senza ricaricarla completamente

- Il W3C ha uno studio in corso per renderlo uno Standard Internet:

  https://www.w3.org/TR/XMLHttpRequest/

INTERFACCIA STANDARD W3C

```
[NoInterfaceObject]
interface XMLHttpRequestEventTarget : EventTarget {// for future use};

[Constructor]
interface XMLHttpRequest : XMLHttpRequestEventTarget {

  // event handler attributes
  attribute Function onreadystatechange;

  // states
  const unsigned short UNSENT = 0;
  const unsigned short OPENED = 1;
  const unsigned short HEADERS_RECEIVED = 2;
  const unsigned short LOADING = 3;
  const unsigned short DONE = 4;
  readonly attribute unsigned short readyState;
```

INTERFACCIA STANDARD W3C

```
// request
 void open(DOMString method, DOMString url);
 void open(DOMString method, DOMString url, boolean async);
 void open(DOMString method, DOMString url, boolean async, DOMString? user);
 void open(DOMString method, DOMString url, boolean async, DOMString? user, DOMString? password);
 void setRequestHeader(DOMString header, DOMString value);
 void send();
 void send(Document data);
 void send([AllowAny] DOMString? data);
 void abort();

 // response
 readonly attribute unsigned short status;
 readonly attribute DOMString statusText;
 DOMString getResponseHeader(DOMString header);
 DOMString getAllResponseHeaders();
 readonly attribute DOMString responseText;
 readonly attribute Document responseXML;                    };
```

CREAZIONE DELL'OGGETTO

- Tutti i browser moderni (IE7+, Firefox, Chrome, Safari e Opera) implementano l'interfaccia W3C con un oggetto nativo ***XMLHttpRequest***

- La sintassi per la creazione è:

```
variable = new XMLHttpRequest();
```

- Mentre le vecchie versioni di Internet Explorer (IE5 e IE6) usano un oggetto ActiveX:

```
variable = new ActiveXObject("Microsoft.XMLHTTP");
```

CREAZIONE DELL'OGGETTO

Per gestire tutti i browser bisogna controllare se il browser supporta l'oggetto ***XMLHttpRequest*** :

```
var xmlhttp;


if (window.XMLHttpRequest)
  {// IE7+, Firefox, Chrome, Opera, Safari
   xmlhttp = new XMLHttpRequest();
  }
else
  {// IE6, IE5
   xmlhttp = new
        ActiveXObject("Microsoft.XMLHTTP");
  }
```

INVIO DELLA RICHIESTA

Per inviare una richiesta al server bisogna usare i metodi `open()` e `send()` dell'oggetto ***XMLHttpRequest***:

`open(method, url, async)`: specifica il tipo di richiesta

- `method`: GET o POST
- `url`: indirizzo server
- `async`: true (asincrona) o false (sincrona)

`send(string)`: invia la richiesta al server

- `string`: solo POST

Esempio

```
xmlhttp.open("GET","ajax_info.txt",true);
xmlhttp.send();
```

OPEN(*METHOD*, . . ., . . .)

- GET è più semplice di POST, e può essere usato nella maggior parte dei casi

Andrebbe usata la richiesta POST quando:

a) Non è possibile utilizzare un file memorizzato nella cache (aggiornamento di un file o di un database sul server), infatti la GET controlla e eventualmente usa il file in cache

b) Bisogna inviare al server una grande quantità di dati (POST non ha limitazioni in dimensione)

c) Bisogna inviare l'input utente

RICHIESTA *GET*

a. Una semplice richiesta GET:

```
xmlhttp.open("GET","demo_get.php",true);
xmlhttp.send();
```

b. Nel precedente esempio potremo ottenere un risultato presente nella cache, per ovviare si può aggiungere un ID univoco all'URL:

```
xmlhttp.open("GET",
             "demo_get.php?t=" + Math.random(),
             true);
xmlhttp.send();
```

RICHIESTA *GET*

c. Se si vuole inviare informazioni con il metodo GET, bisogna aggiungere informazioni all'URL:

```
xmlhttp.open("GET",
        "demo_get2.php?fname=Henry&lname=Ford",
         true);
xmlhttp.send();
```

RICHIESTA *POST*

a. Una semplice richiesta POST:

```
xmlhttp.open("POST","demo_post.php",true);

xmlhttp.send();
```

b. Per fare una POST dei dati, come una form HTML, si deve aggiunge un header HTTP con il metodo `setRequestHeader()` e specificare nel metodo `send()` il dato che vuole inviare:

```
xmlhttp.open("POST","ajax_test.php",true);

xmlhttp.setRequestHeader("Content-type",
        "application/x-www-form-urlencoded");

xmlhttp.send("fname=Henry&lname=Ford");
```

`setRequestHeader(header, value)`: aggiunge alla richiesta HTTP un nome e un valore di un header

OPEN(..., *URL*, ...)

- Il parametro `url` specifica l'indirizzo di un file sul server:

```
xmlhttp.open("GET", "ajax_test.php", true);
```

- Il file può essere di tipo testuale (.txt e .xml), oppure un componente software lato server (.aspx, .php, .jsp, ...) che elabora la richiesta e poi risponde

OPEN(..., ..., *ASYNC*)

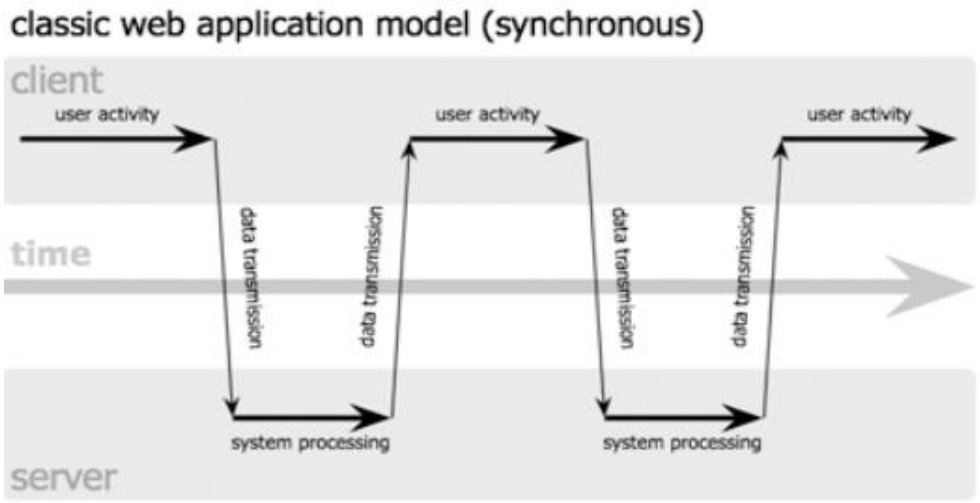
- Per parlare di Ajax il parametro `async` deve essere impostato a `true`:

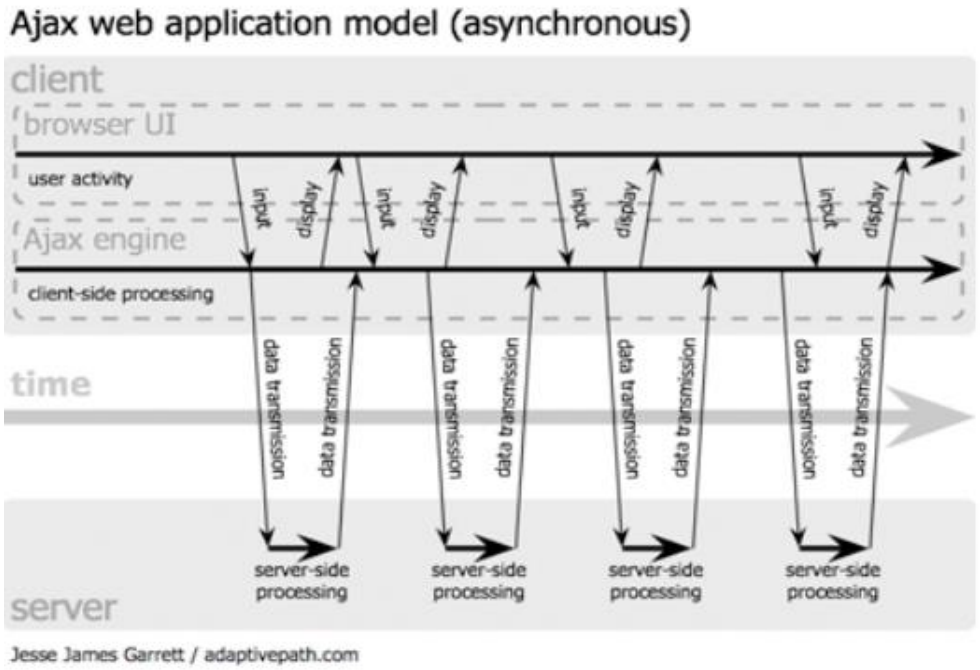
```
xmlhttp.open("GET", "ajax_test.php", true);
```

- L'invio di richieste **asincrone** offre un miglioramento delle prestazioni perché gli script non aspettano la risposta del server, fanno altro per poi riprendere quando questa arriva

- Quando un server è sovraccarico e risponde con ritardo, le richieste **sincrone** possono causare rallentamenti o fermi della applicazione

RICHIESTE SINCRONE E ASINCRONE





RICHIESTA SINCRONA

- L'uso di `async = false` non è raccomandato, può andar bene per piccole quantità di dati

- Quando si usa `async = false`, non si deve implementare la funzione collegata al gestore di eventi `onreadystatechange`, ma basta inserire del codice dopo l'istruzione `send()`

Esempio

```
xmlhttp.open("GET", "ajax_info.txt", false);

xmlhttp.send();

document.getElementById("myDiv").innerHTML = xmlhttp.responseText;
```

RICHIESTA ASINCRONA

- Quando si usa `async = true`, si deve implementare la funzione collegata al gestore di eventi `onreadystatechange` da eseguire quando la risposta è pronta

Esempio

```
xmlhttp.onreadystatechange = function()
  {
  if(xmlhttp.readyState == 4 && xmlhttp.status == 200)
    {
       document.getElementById("myDiv").innerHTML
                                = xmlhttp.responseText;
    }
  }
xmlhttp.open("GET", "ajax_info.txt", true);
xmlhttp.send();
```

RISPOSTA DEL SERVER

- Le **proprietà** per ottenere la risposta del server sono:
  1. `responseText`
  2. `responseXML`

- Se la risposta non è XML, usare la proprietà `responseText` che restituisce una stringa

Esempio

```
document.getElementById("myDiv").innerHTML = xmlhttp.responseText;
```

RISPOSTA DEL SERVER

- Se la risposta del server è XML e si vuole fare il parse, usare la proprietà `responseXML`

Esempio

```
xmlDoc = xmlhttp.responseXML;
txt = "";
x = xmlDoc.getElementsByTagName("ARTIST");

for (i=0; i < x.length; i++)
{
  txt = txt + x[i].childNodes[0].nodeValue
              + "<br />";
}

document.getElementById("myDiv").innerHTML = txt;
```

PROPRIETÀ

- `readyState`: contiene lo **stato di XMLHttpRequest**

  `0`: **richiesta non inizializzata** (`UNSENT`)
  `1`: **connessione al server stabilita** (`OPENED`)
  `2`: **richiesta ricevuta** (`HEADERS_RECEIVED`)
  `3`: **elaborazione richiesta in corso** (`LOADING`)
  `4`: **richiesta completa e risposta pronta** (`DONE`)

- `status`: contiene il **codice di stato HTTP**

  `0`: **se lo stato è** `UNSENT` **o** `OPENED`
  `200`: **ok**
  `404`: **pagina non trovata**

EVENTI

- Quando è inviata una richiesta ad un server e si vuole fare una qualche azione basata sulla risposta, si utilizza l'evento collegato a `onreadystatechange` dell'oggetto XMLHttpRequest

- La funzione collegata al gestore di eventi `onreadystatechange` è **invocata automaticamente ad ogni cambiamento della proprietà** `readyState`, essa specifica cosa accadrà quando la risposta del server è pronta

ESEMPIO EVENTI

```
xmlhttp.onreadystatechange = function()
  {
  if (xmlhttp.readyState == 4 // Stato XMLHttpRequest: DONE
       &&
       xmlhttp.status == 200) // Stato HTTP: OK
    {
       document.getElementById("myDiv").innerHTML
              = xmlhttp.responseText;
    }
  }
```

L'evento `onreadystatechange` **è scatenato 4 volte, una volta per ogni cambio dello stato di XMLHttpRequest** (`readyState`)

ARGOMENTI

- Introduzione
- XMLHttpRequest
- JSON
- Framework Ajax

JSON

- **JavaScript Object Notation** è un semplice formato per lo scambio dei dati, insieme all'XML, è il formato più comune per la trasmissione dei dati su canale HTTP da parte delle applicazioni Ajax.

- Come per l'XML, JSON contiene sia dato sia etichetta permettendo la creazione di strutture dati autodescrittive

- Si basa su un sottoinsieme del linguaggio di programmazione JavaScript (Standard ECMA-262 terza edizione)

https://ecma-international.org/wp-content/uploads/ECMA-262_14th_edition_june_2023.pdf

JSON

- E' un **formato testuale** completamente **indipendente dal linguaggio di programmazione**, esistono librerie JSON per la maggior parte dei linguaggi (C, C++, Java, Python, Perl, PHP).

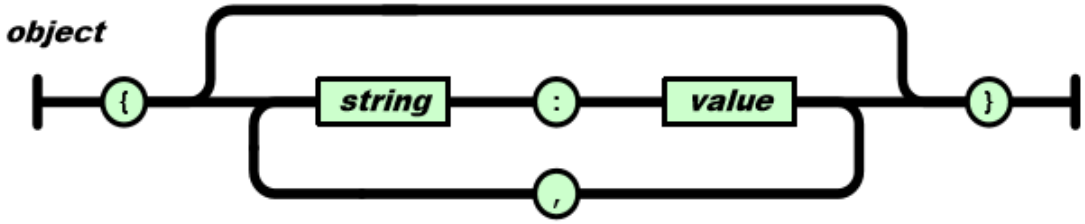
- Definisce un piccolo insieme di regole di formattazione per la rappresentazione di strutture dati portabili, ed è basato sul concetto di **oggetto JavaScript** e di **array associativo** (array i cui indici sono parole).

TIPI DI DATO JSON

Tipo	Descrizione
Number	***numero*** in uno dei formati JavaScript
String	**raccolta** di **caratteri Unicode** (***stringa***)
Boolean	***vero*** o ***falso***
Array	**raccolta** ordinata di ***valori***
Value	può essere una ***stringa***, un ***numero***, ***vero*** o ***falso***, ***nullo***, un ***oggetto*** o un ***array***
Object	**serie** non ordinata di ***nomi/valori***

JSON OBJECT

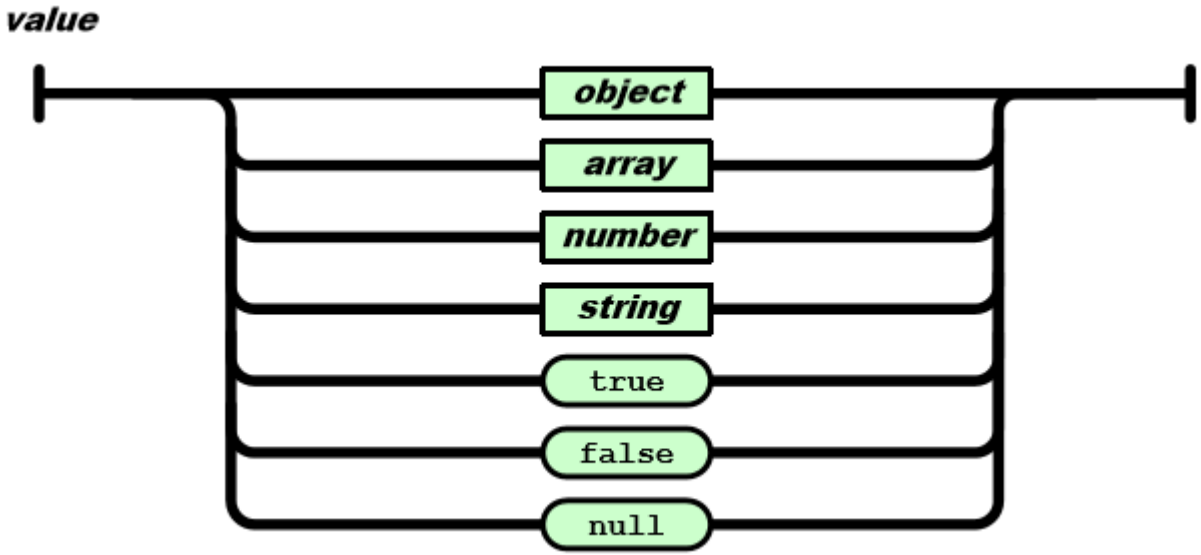
- E' una **serie** non ordinata di ***nomi/valori***
- Inizia con **{** e finisce con **}**
- Ogni nome è seguito da **:**
- La coppia di ***nome/valore*** sono separata da **,**



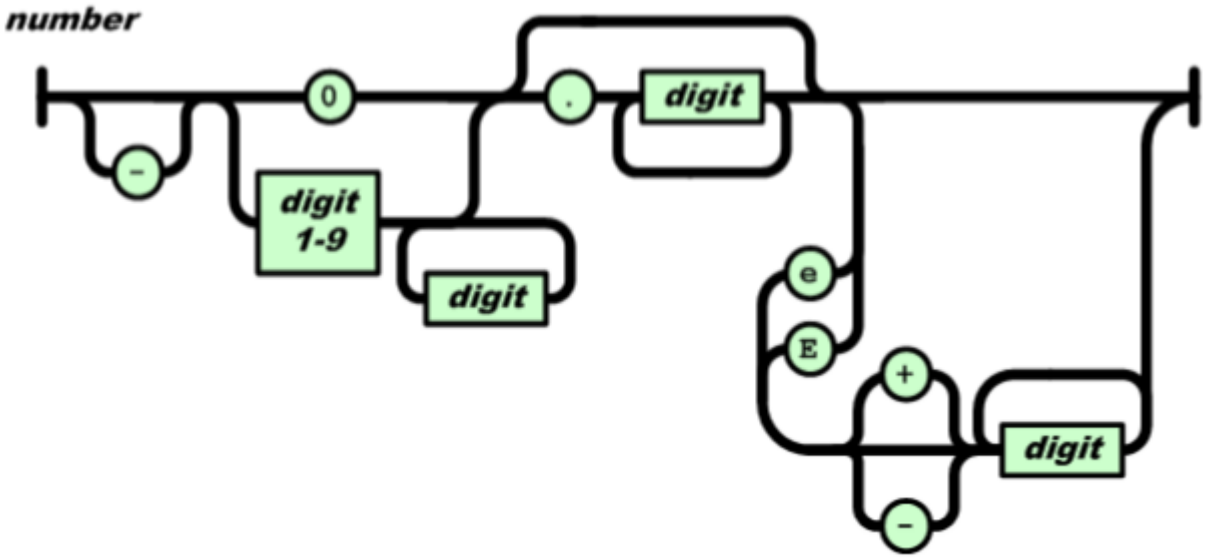

```
{
    "id": "011A",
    "language": "JAVA",
    "price": 500,
}
```

JSON VALUE

- Può essere una ***stringa*** tra virgolette, o un ***numero***, o ***vero*** o ***falso*** o ***nullo***, o un ***oggetto*** o un ***array***
- Queste strutture possono essere annidate

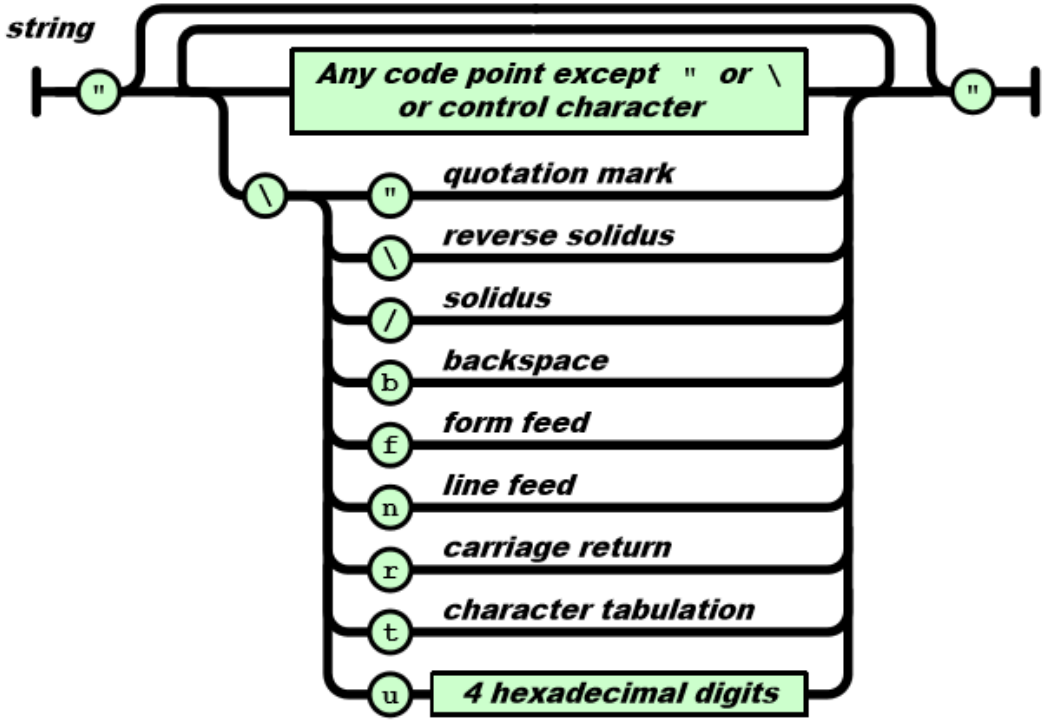


JSON NUMBER



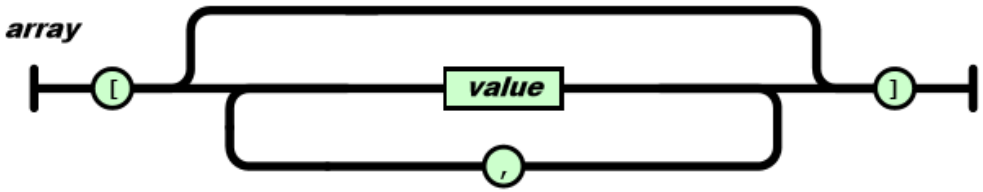
JSON STRING

- E' una **raccolta** di **caratteri Unicode** tra virgolette
- Per le **sequenze di escape** utilizza la **barra rovesciata**
- Un carattere è una stringa di lunghezza uno



JSON ARRAY

- E' una **raccolta** ordinata di ***valori***
- Comincia con **[** e finisce con **]**
- I valori sono separati da **,**




```
{
   "book": [

      {
         "id":"01",
         "language": "Java",
         "edition": "third",
         "author": "Herbert Schildt",
      },

      {
         "id":"07",
         "language": "C++",
         "edition": "second",
         "author": "E.Balagurusamy",
      }

   ]
}
```

JSON CON PHP

- PHP versione 5.2.0 mette a disposizione delle funzioni per lavorare con JSON

Funzioni	Librerie
json_encode	Restituisce la rappresentazione JSON di un valore
json_decode	Decodifica una stringa JSON
json_last_error	Restituisce l'ultimo errore avvenuto

FUNZIONE JSON_ENCODE

- **Sintassi**

```
string json_encode ( $value [, $options = 0 ] )
```

- **Parametri**

  `value` – **Il valore da codificare, utilizza solo la codifica UTF-8**

  `options` – **facoltative, i valori possibili sono:**

  `JSON_HEX_QUOT, JSON_HEX_TAG, JSON_HEX_AMP, JSON_HEX_APOS,`
  `JSON_NUMERIC_CHECK, JSON_PRETTY_PRINT,`
  `JSON_UNESCAPED_SLASHES, JSON_FORCE_OBJECT`

ESEMPI JSON_ENCODE

```php
<?php
   $arr = array('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4, 'e' => 5);
   echo json_encode($arr);
?>
```

```
{"a":1,"b":2,"c":3,"d":4,"e":5}
```

```php
<?php
   class Emp {
      public $name = "";
      public $hobbies  = "";
      public $birthdate = "";
   }

   $e = new Emp();
   $e->name = "sachin";
   $e->hobbies  = "sports";
   $e->birthdate = date('m/d/Y h:i:s a', "8/5/1974 12:20:03 p");
   $e->birthdate = date('m/d/Y h:i:s a', strtotime("8/5/1974 12:20:03"));

   echo json_encode($e);
?>
```

```
{"name":"sachin","hobbies":"sports","birthdate":"08\/05\/1974 12:20:03 pm"}
```

FUNZIONE JSON_DECODE

- **Sintassi**

```
mixed json_decode ($json_string [,$assoc = false

[, $depth = 512 [, $options = 0 ]]])
```

- **Parametri**

`json_string` – la stringa json codificata UTF-8

`assoc` – è un booleano, quando è `TRUE` restituisce gli oggetti convertiti in un array associativo

`depth` – è un intero che specifica la profondità della ricorsione

`options` – è un intero di JSON decode, `JSON_BIGINT_AS_STRING` è supportato

ESEMPIO JSON_DECODE

```php
<?php
   $json = '{"a":1,"b":2,"c":3,"d":4,"e":5}';

   var_dump(json_decode($json));
   var_dump(json_decode($json, true));
?>
```

```
object(stdClass)#1 (5) {
   ["a"] => int(1)
   ["b"] => int(2)
   ["c"] => int(3)
   ["d"] => int(4)
   ["e"] => int(5)
}


array(5) {
   ["a"] => int(1)
   ["b"] => int(2)
   ["c"] => int(3)
   ["d"] => int(4)
   ["e"] => int(5)
}
```

ARGOMENTI

- Introduzione
- XMLHttpRequest
- JSON
- Framework Ajax

FRAMEWORK AJAX

Sono librerie JavaScript che semplificano la creazione di applicazioni che implementano Ajax, i cui scopi fondamentali sono:

1. **Astrazione**: gestiscono le differenze tra un browser e l'altro e forniscono un modello unico di programmazione compatibile con molti browser

2. **Struttura**: forniscono un modello omogeneo di progetto dell'applicazione, indicando con esattezza dove e come inserire le caratteristiche specifiche dell'applicazione

3. **Libreria di widget**: forniscono una ricca collezione di componenti di presentazione assemblabili per creare velocemente interfacce sofisticate e modulari

FRAMEWORK AJAX

Librerie JavaScript

- jQuery (https://jquery.com/)
- Prototype (http://www.prototypejs.org/)

JavaScript Framework

- AngularJS (https://angularjs.org/)
- React (https://facebook.github.io/react/)
- Backbone (http://backbonejs.org/)
- Ember (http://emberjs.com/)

RISORSE

- https://it.wikipedia.org/wiki/AJAX
- https://www.w3schools.com/js/js_ajax_intro.asp
- https://www.ietf.org/rfc/rfc4627.txt?number=4627
- https://www.w3schools.com/jsref/jsref_obj_json.asp
- https://alexbosworth.backpackit.com/pub/67688
- https://developer.mozilla.org/en/AJAX/Getting_Started
- https://www.w3.org/TR/XMLHttpRequest/
- **Pro Apache Struts with Ajax** by John Carnell, Rob Harrop, Kunal Mittal (2006) (CHAPTER 12 - Struts and Ajax)
- https://www.json.org/json-it.html
- https://ecma-international.org/wp-content/uploads/ECMA-262_14th_edition_june_2023.pdf