

Client side web development: javascript e tecnologia AJAX



Creative Commons License

Attribution-NonCommercial-ShareAlike 2.5

<http://creativecommons.org/licenses/by-nc-sa/2.5/>

claudio@cicali.org

http://claudio.cicali.org/files/cicali_javascript.pdf

Web design, ovvero:

- **Struttura**
- **Presentazione**
- **Comportamento**

Web design, ovvero:

- Struttura ✓ (X)HTML
- Presentazione ✓ CSS
- **Comportamento** ✓ **javascript**

Obiettivi

- **Imparare le basi del linguaggio javascript**

Obiettivi

- Imparare le basi del linguaggio javascript
- **Imparare ad interagire con un documento HTML**

Obiettivi

- Imparare le basi del linguaggio javascript
- Imparare ad interagire con un documento HTML
- **Comunicare con un server**

Obiettivi

- Imparare le basi del linguaggio javascript
- Imparare ad interagire con un documento HTML
- Comunicare con un server
- **Creare RCA con AJAX**



Il linguaggio javascript

javascript è un linguaggio di programmazione

Linguaggio di scripting

Gira all'interno del browser (host environment)

javascript è un linguaggio strano

Non esistono array associativi
ma gli oggetti sono array associativi

Non esistono metodi
ma ogni funzione è un metodo

history.back: ~1997

window.open days

history.forward: ~2003

document.getElementById days

XMLHttpRequest days

Uso

Direttamente nel sorgente HTML di un documento

```
...  
<script type="text/javascript">  
    alert("hello world!")  
</script>  
...
```

All'interno di un file esterno al documento

```
...  
<script type="text/javascript" src="tools.js"></script>  
<script type="text/javascript" src="editor.js"></script>  
...
```

Uso

Il codice javascript può trovarsi in qualsiasi zona del documento, anche alla fine, dopo il `<body>`

```
...  
<p>Distinti saluti!</p>  
<script type="text/javascript">  
    alert("Arrivederci")  
</script>  
</body>  
</html>
```

Uso

L'attributo **language** dell'elemento script non è normalmente necessario: serve per richiedere il supporto della **versione minima** del linguaggio

```
<script type="text/javascript" language="javascript1.4">  
    alert("Un messaggio solo per gli utenti con almeno il js 1.4")  
</script>
```

Uso

Del codice javascript può essere inserito direttamente come valore degli attributi che specificano un evento

```
<input type="button" onclick="alert('Ouch!'); return false;">
```

<noscript>

```
<noscript>  
Il tuo browser non supporta javascript  
o questo è stato disattivato  
</noscript>
```

Le versioni del linguaggio

Creato da **Netscape**

1995: javascript **1.0**: Netscape 1.0, IE 3

...

2005: javascript **1.5** (IE 6, Mozilla...)

La versione del javascript non è molto importante

E' più importante **la versione del browser**

Ecma, javascript, javascript: one face one race

Netscape crea il linguaggio JavaScript

Microsoft ribattezza "il suo" javascript come **JScript**

ECMA standardizza il linguaggio (da lì, ECMAScript)
<http://www.ecma-international.org/>

Gli sviluppi continuano, convergendo
tutti verso gli standard ECMA (ECMA 262)
La versione corrente per ECMAScript è la **3**

Javascript 2.0, ECMA 4

I commenti

```
/* Commento  
Multilinea */
```

```
// Commento singola linea
```

```
/*  
  Funzioni di generica utilità  
  Estensione dell'oggetto Number  
*/  
Object.extend(Number.prototype, {  
  // Ritorna il numero successivo  
  succ: function() {  
    return this + 1;  
  },  
  
  // Simula un iteratore  
  times: function(iterator) {  
    $R(0, this, true).each(iterator);  
    return this;  
  }  
});
```

Il linguaggio: tipi di dato

- Object (e Array)
- String
- Number
- Boolean
- Null
- Undefined

Il linguaggio: oggetti predefiniti

- Boolean
- String
- Number
- Function
- Date
- Math
- RegExp
- *Errors (exceptions)

typeof e instanceof

Per conoscere il tipo di una variabile si usa **typeof**

Spesso utile per verificare lo stato "Undefined"

```
if (typeof variabile_sconosciuta == "Undefined")  
    alert("Variabile non impostata")
```

Per conoscere il tipo di oggetto, si effettua il test con **instanceof**

```
if (presunto_array instanceof Array)  
    alert("La variabile è un array")
```

Le variabili

```
var nome_variabile
```

“var” non è obbligatorio, ma occhio allo *scope*

javascript è **case sensitive**

Scope delle variabili

Scope:

- Locale (alle funzioni)
- Globale (all'interno dell'host environment)

Il linguaggio

javascript comprende naturalmente tutti i più conosciuti costrutti e operatori

```
if ... else ...  
while  
for  
do ... while  
switch  
break, continue
```

Gli Array

```
marche = new Array("Mercedes", "Ford", "BMW")  
alert(marche[0])
```

Gli array possono contenere elementi di qualsiasi tipo

```
container = new Array(7, "stringa", new Object())
```

... e dunque:

```
multidim = new Array(1, new Array(1,2,3))
```

Array: quanti elementi?

```
marche = ["Mercedes", "Ford", "BMW"]  
marche.length
```

Array: notazione compatta

```
marche = ["Mercedes", "Ford", "BMW"]
```

```
primi_numeri_primi = [1, 3, 5, 7]
```

Array: aggiungere e togliere elementi

```
marche = ["Mercedes", "Ford", "BMW"]  
marche[54390] = "Fiat" // crea anche 54387 elementi vuoti
```

```
marche = ["Mercedes", "Ford", "BMW"]  
marche.push("Fiat") // Aggiunge un elemento in fondo
```

```
marche = ["Mercedes", "Ford", "BMW"]  
marche.pop() // Toglie (e ritorna) l'ultimo elemento
```

Array associativi? Hash?

Non esistono specifiche strutture di questo tipo

Si possono simulare

Funzioni anonime

Le funzioni sono **oggetti**

In questo senso è possibile usare le funzioni *anonime*

```
mia_funzione = function (a)
{
    return a * 2
}

alert(mia_funzione(8))
```

Funzioni anonime

Le funzioni sono **oggetti**

```
var f1 = new Function(["v"], "return v * 16")
alert(f1.call(this, 3))

var f2 = new Function("return arguments[0] * 16")
alert(f2(3))

var f3 = function(v) {
    return v * 16
}
alert(f3(3))

function f4(v) {
    return v * 16
}
alert(f4(3))
```

Le funzioni

Passaggio per reference degli oggetti e
per valore delle proprietà

Per reference

```
function modify(param)
{
    param.altra_var += " modified"
}

var mia_var = new String("Ciao")

mia_var.altra_var = "Mondo"

modify(mia_var)

// Adesso mia_var.altra_var
// vale "Mondo modified"
```

Per valore

```
function modify(param)
{
    param += " modified"
}

var mia_var = new String("Ciao")

mia_var.altra_var = "Mondo"

modify(mia_var.altra_var)

// mia_var.altra_var vale
// ancora "Mondo"
```

Le funzioni (2)

Tutte le funzioni sono a numero di argomenti variabili

Si accede alla lista degli argomenti
tramite l'array *arguments*

```
function senza_parametri()  
{  
    return arguments[0]  
}  
  
alert(senza_parametri('salve'))
```

Per questo motivo non esiste il concetto di *overload*

Gli oggetti – creazione

Si possono creare oggetti tramite la notazione compatta

```
mio_fratello = {  
  nome: 'Alberto',  
  cognome: 'Cicali',  
  eta: '35',  
  say_hello: function() {  
    return this.nome + " " + this.cognome + " ti saluta!"  
  },  
  automobili: ['fiat punto', 'fiat bravo']  
}
```

Gli oggetti - creazione

Definire il costruttore è sufficiente per creare un nuovo oggetto

```
function persona(nome, cognome, eta)
{
    this.nome = nome
    this.cognome = cognome
    this.eta = eta
}

var mio_fratello = new persona('alberto', 'cicali', 35)

alert(mio_fratello.nome)
```

Gli oggetti – creazione

Se si necessita di una sola istanza “statica” di un oggetto si può usare Object

```
mio_fratello = new Object()  
mio_fratello.nome = 'alberto'  
mio_fratello.cognome = 'cicali'  
mio_fratello.eta = 35  
  
alert(mio_fratello.eta)
```

Gli oggetti – creazione

Per i metodi è comodo usare le *funzioni anonime*

```
function persona(nome, cognome, eta)
{
    this.nome = nome
    this.cognome = cognome
    this.eta = eta
    this.say_hello = function() {
        return this.nome + " " + this.cognome + " ti saluta!"
    }
}

var mio_fratello = new persona('alberto', 'cicali', 35)

alert(mio_fratello.say_hello())
```

Ma i metodi in realtà **non esistono**

```
mio_fratello.say_hello()
// è equivalente a
(mio_fratello.say_hello).call()
```

Gli oggetti

Si può accedere alle proprietà tramite la notazione `."` o con la stessa notazione degli `"array associativi"`

```
var test = "Ciao"

alert(test.length)

alert(test['length'])
```

Questa caratteristica ci permette di simulare gli array associativi

```
var assoc = new Object()
assoc['mario'] = 47
assoc['bruno'] = 32
assoc['carla'] = 21
```

Gli oggetti: estensione

Attributi e metodi degli oggetti possono essere aggiunti quando si vuole

```
var person = new Object
person.name = "Claudio"

function get_age()
{
    return this.age
}

person.get_age = get_age
alert(person.get_age)
```

```
var stringa = new String("Salve")
stringa.autore = "Claudio"
```

Gli oggetti: estensione (2)

Può essere modificato anche il *prototipo* dell'oggetto base

```
function add_something()  
{  
    return this + "something"  
}  
  
String.prototype.add_something = add_something  
  
var test = "Claudio"  
  
alert(test.add_something())
```

Gli oggetti: inspecting

Si usa la parola chiave "in" per verificare la presenza di una certa proprietà

```
mio_fratello = new Object()
mio_fratello.nome = 'alberto'

if ("nome" in mio_fratello)
    alert("mio_fratello ha un nome")
```

Si può dare un'occhiata all'interno di un oggetto con un ciclo for

```
for (nome_attributo in mio_fratello)
{
    alert(mio_fratello[nome_attributo])
}
```

Ereditarietà?

Non esiste un vero e proprio concetto di **ereditarietà**

Una gerarchia di oggetti la si crea definendo una classe come *prototype* di un'altra

```
function Employee ()
{
  this.name = "";
  this.dept = "general";
}

function WorkerBee ()
{
  this.projects = [];
}
WorkerBee.prototype = new Employee;

function Engineer ()
{
  this.dept = "engineering";
  this.machine = "";
}
Engineer.prototype = new WorkerBee;
```

Gestione delle eccezioni

Try, catch, throw & finally

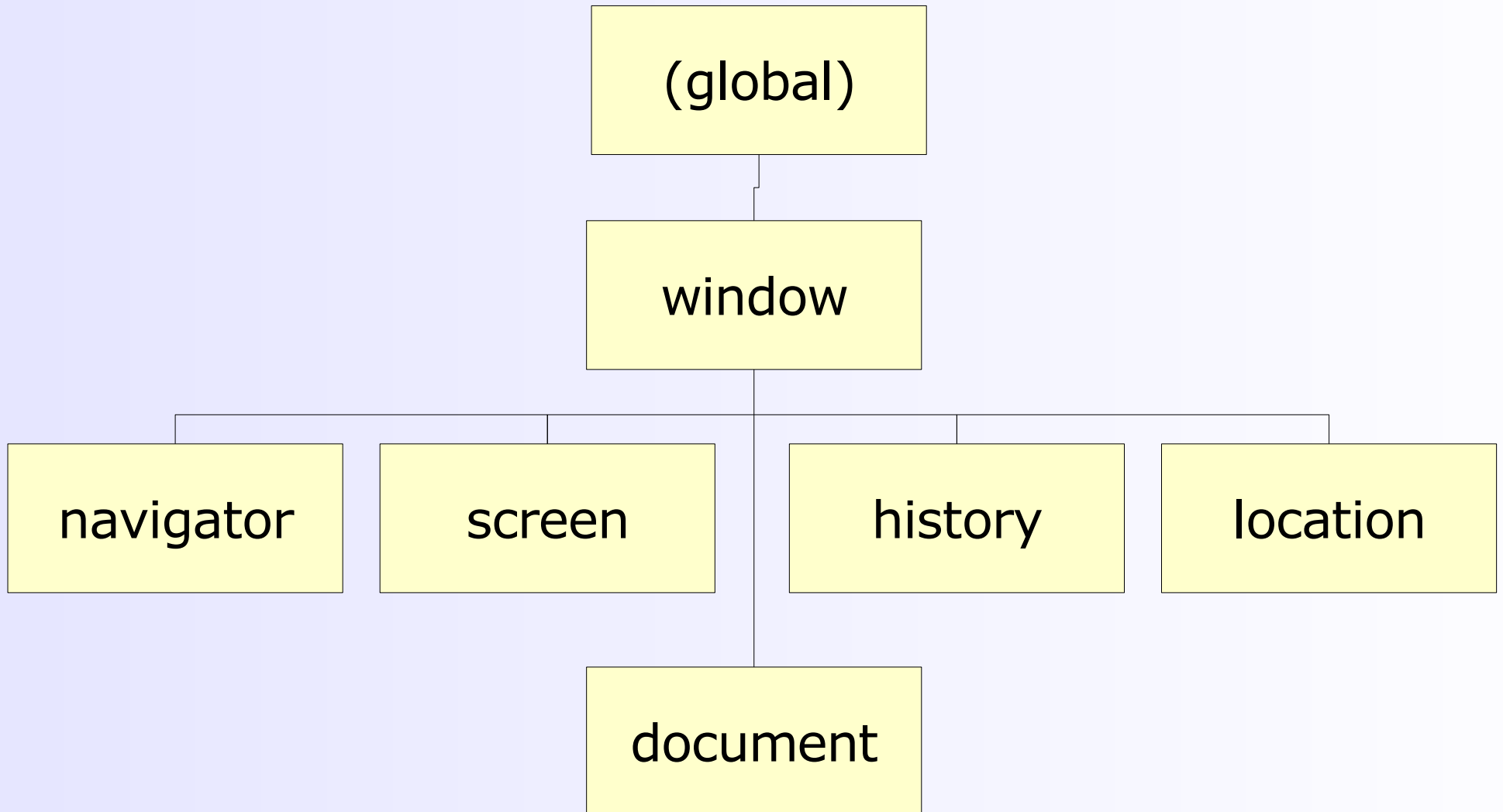
```
try {  
    funzione_inesistente()  
}  
catch(e) {  
    alert("Errore di tipo" + e.name)  
    alert("Messaggio" + e.message)  
}  
finally {  
    alert("Fine")  
}
```

"e" è istanza dell'oggetto Error

The image features a green background with three concentric circles. The innermost circle is a dark green, the middle one is a medium green, and the outermost one is a lighter green. The text "Interazione con il browser" is centered horizontally and partially overlaps the inner and middle circles.

Interazione con il browser

Gli oggetti predefiniti di più alto livello



Oggetto WINDOW

E' presente anche senza un documento

Tramite **window** è possibile modificare il *chrome* della finestra

self è il sinonimo di window

WINDOW

Tramite **window** è possibile aprire nuove finestre

```
var nuova_finestra = window.open(URL, titolo, opzioni)
```

Si ottiene così una reference alla finestra appena creata

```
nuova_finestra.close()
```

WINDOW

Metodi e proprietà di comune utilizzo:

```
window.alert()  
  
var confirmed = window.confirm("Sei sicuro?")  
  
var nome = window.prompt("Inserisci il tuo nome")  
  
window.status="Sono qui per dare fastidio"
```

I metodi della finestra principale possono essere usati senza esplicito riferimento all'oggetto (window)

```
alert()  
  
var confirmed = confirm("Sei sicuro?")  
  
var nome = prompt("Inserisci il tuo nome")  
  
status="Sono qui per dare fastidio"
```

Oggetto NAVIGATOR

Contiene le proprietà relative al browser

```
// Alcune delle proprietà dell'oggetto navigator  
navigator.appCodeName // "code name" del browser.  
navigator.appName // Nome del browser.  
navigator.appVersion // Versione del browser.  
navigator.userAgent // User Agent.
```

Oggetto SCREEN

Permette di accedere alle proprietà fisiche dello "schermo"

```
screen.width  
screen.height  
screen.availWidth  
screen.availHeight
```

Oggetto HISTORY

Metodi e proprietà per accedere (o usare) la lista degli ultimi documenti visitati

L'uso più comune:

```
// Simila la pressione del bottone "Back" dell'UI  
history.go(-1)  
// oppure  
history.back()
```

Oggetto LOCATION

Contiene l'URI correntemente caricato dal browser
(e tutte le sue parti **host**, **pathname** e **port**)

E' possibile modificare direttamente **location.href**

```
location.href = "http://www.google.com"
```

Metodi interessanti sono il **.reload** e il **.replace**

Oggetto DOCUMENT

L'oggetto "principe" del DHTML: contiene il documento corrente e tutti i metodi e proprietà per poterlo ispezionare o modificare

```
// Contiene le seguentiHTMLCollection,  
// retaggio del passato  
  
    forms  
    images  
    applets  
    links  
    anchors  
    embeds  
    plugins
```

document.write è il metodo che si utilizza per scrivere HTML durante il caricamento della pagina



Document **O**bject **M**odel

DOM

API per accedere ai documenti XML e HTML

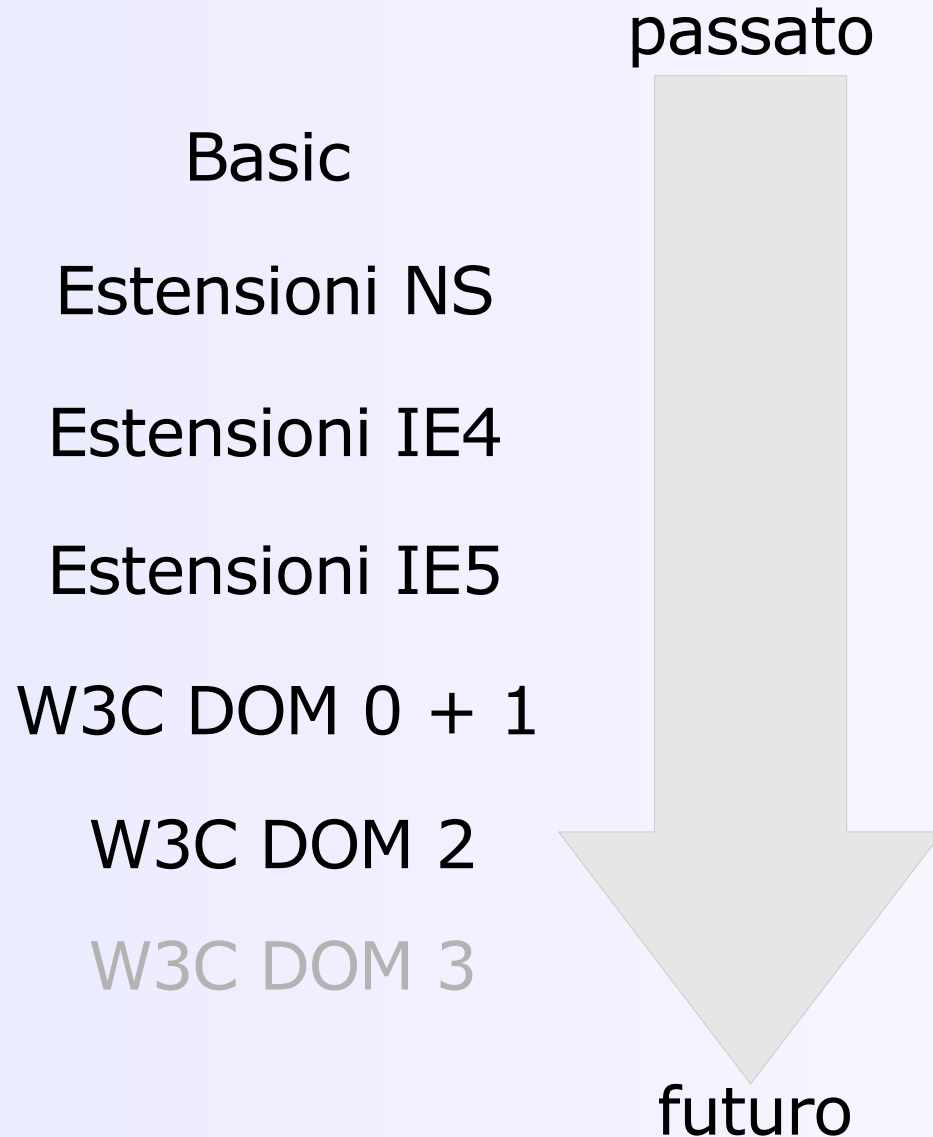
```
interface HTMLCollection {
    readonly attribute unsigned long          length;
    Node                               item(in unsigned long index);
    Node                               namedItem(in DOMString name);
};
```

“Collections in the HTML DOM are assumed to be live meaning that they are automatically updated when the underlying document is changed. ”

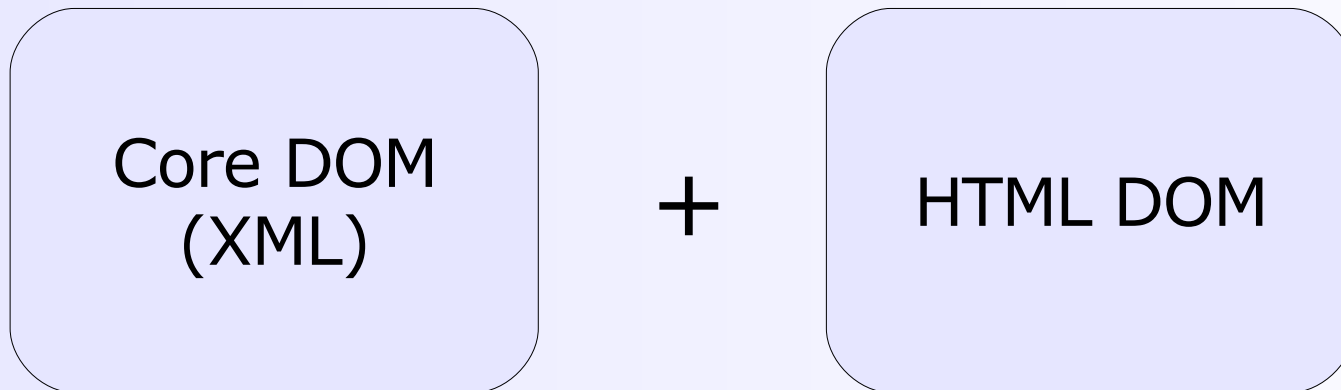
```
interface HTMLInputElement : Element {
    attribute DOMString          id;
    attribute DOMString          title;
    attribute DOMString          lang;
    attribute DOMString          dir;
    attribute DOMString          className;
};
```

<http://www.w3.org/TR/REC-DOM-Level-1/level-one-html.html>

DOM, passato e presente



W3C DOM



Nodi

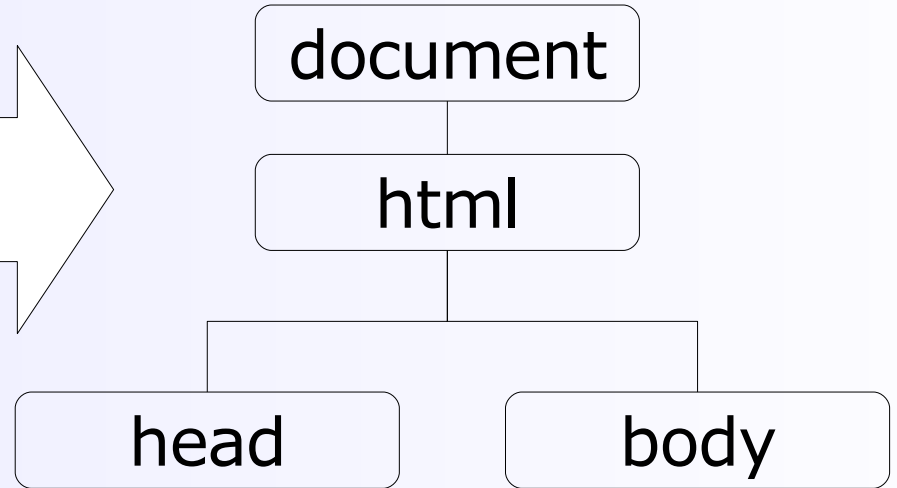
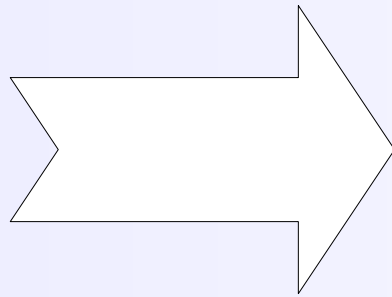
Ogni elemento del DOM è un **nodo**

Il DOM W3C definisce 12 tipi di nodi diversi

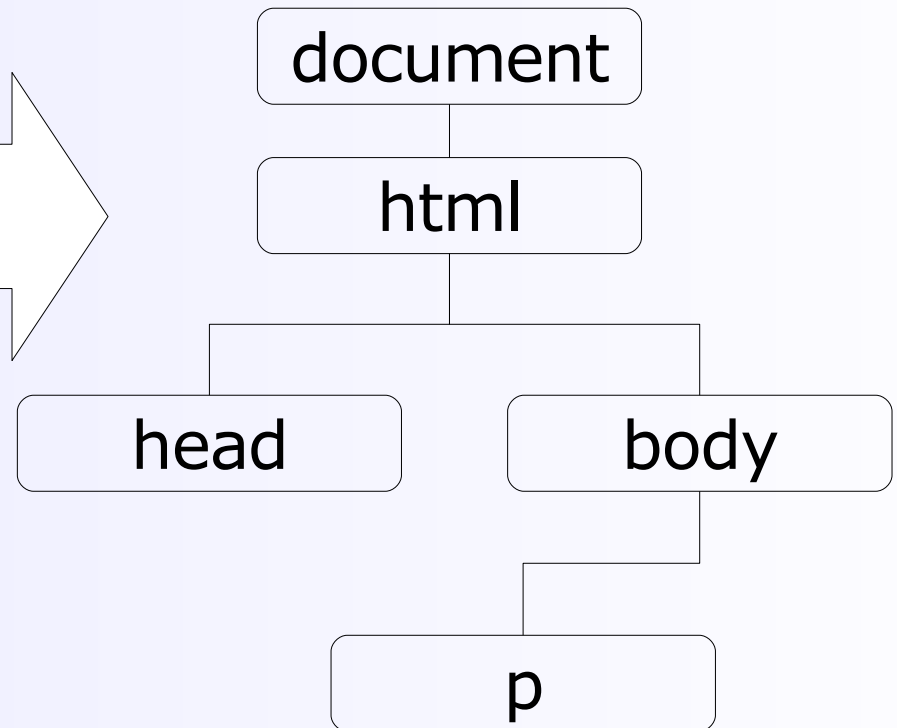
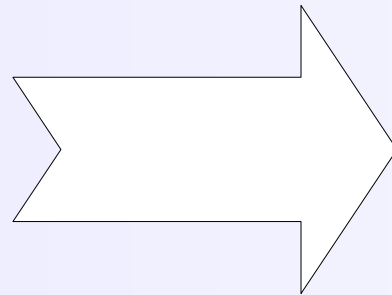
```
interface Node {  
  [...]  
  // NodeType  
  const unsigned short      ELEMENT_NODE          = 1;  
  const unsigned short      ATTRIBUTE_NODE          = 2;  
  const unsigned short      TEXT_NODE              = 3;  
  const unsigned short      CDATA_SECTION_NODE     = 4;  
  const unsigned short      ENTITY_REFERENCE_NODE  = 5;  
  const unsigned short      ENTITY_NODE           = 6;  
  const unsigned short      PROCESSING_INSTRUCTION_NODE = 7;  
  const unsigned short      COMMENT_NODE          = 8;  
  const unsigned short      DOCUMENT_NODE         = 9;  
  const unsigned short      DOCUMENT_TYPE_NODE    = 10;  
  const unsigned short      DOCUMENT_FRAGMENT_NODE = 11;  
  const unsigned short      NOTATION_NODE         = 12;  
  [...]  
};
```

Quelli che più ci interessano sono solo 2:
element node e text node

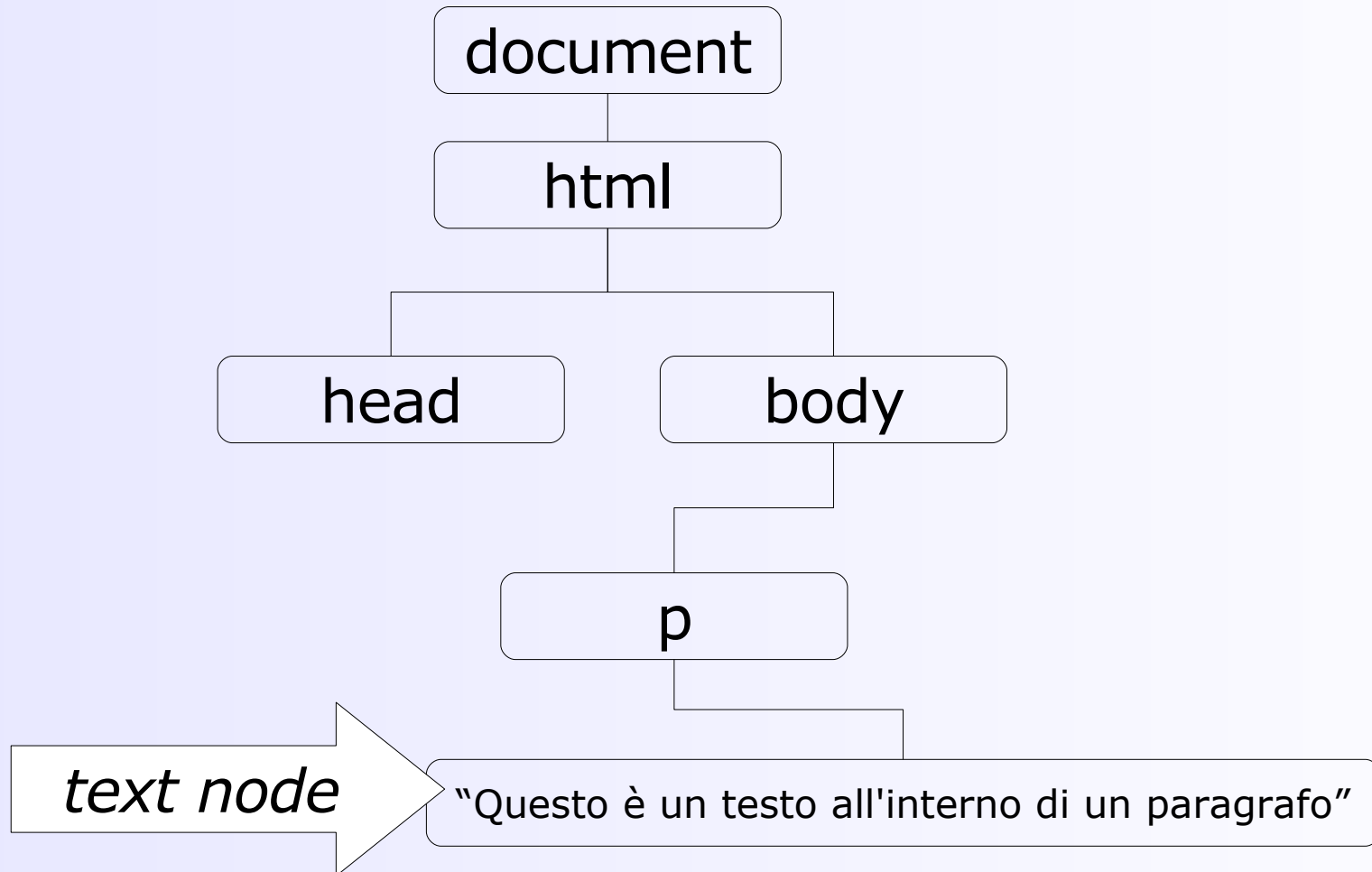
```
<html>
  <head>
</head>
  <body>
</body>
</html>
```



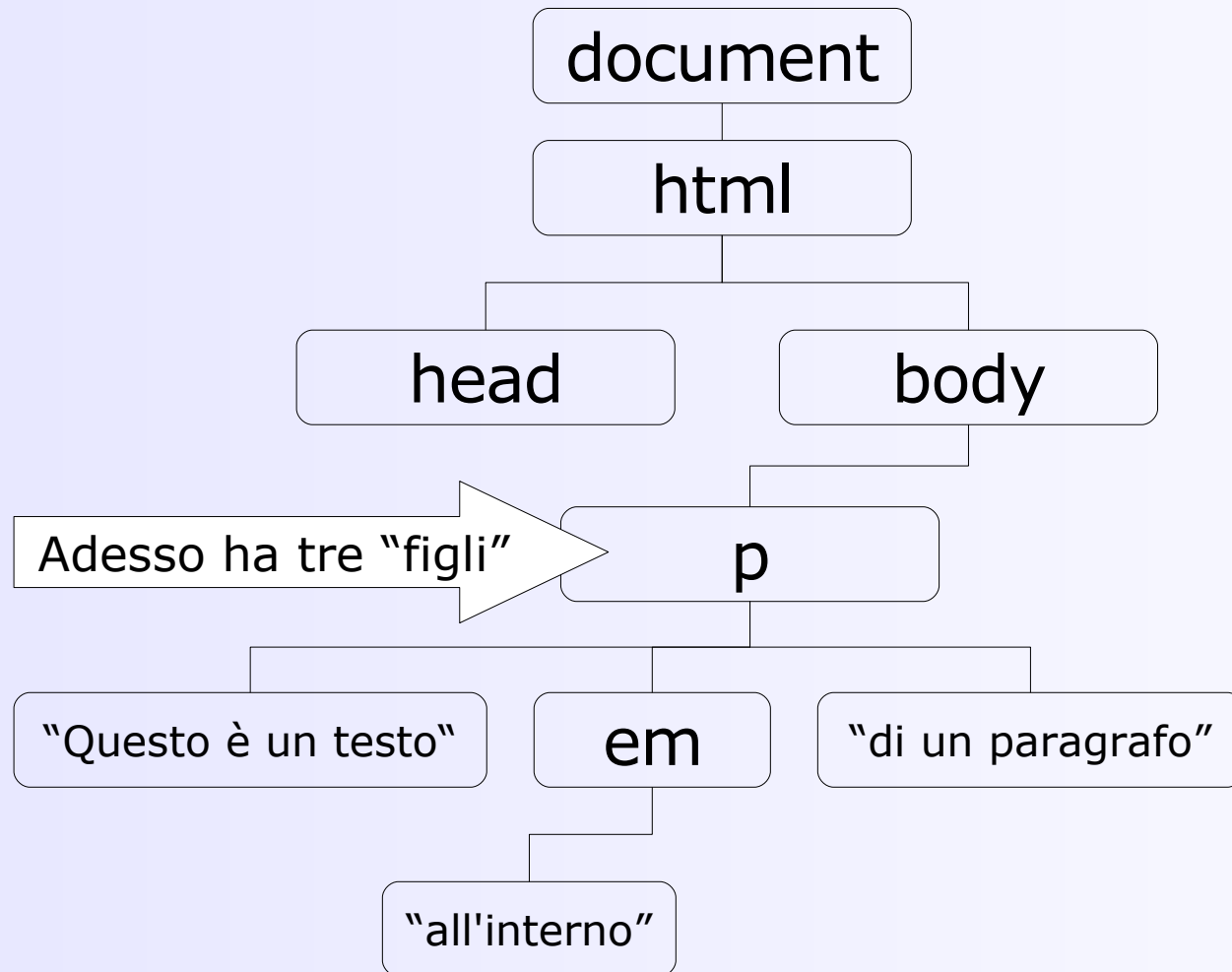
```
<html>
  <head>
</head>
  <body>
    <p>
    </p>
  </body>
</html>
```



```
<html>
  <head>
</head>
  <body>
    <p>Questo è un testo all'interno di un paragrafo</p>
  </body>
</html>
```



```
<html>
  <head>
</head>
  <body>
    <p>Questo è un testo <em>all'interno</em> di un paragrafo</p>
  </body>
</html>
```



Accedere ad un elemento del DOM: l'attributo ID

Introdotta da HTML 4

```
<p id="footer">Distinti saluti</p>
```

Internet Explorer
document.all.id_oggetto

(W3C) DOM Level 0

document.nome_oggetto
oppure document.forms

W3C DOM Level 1

document.getElementById(id_oggetto)

Caratteristiche di ogni nodo

Ogni nodo, come oggetto javascript, possiede **proprietà** e metodi

Le proprietà:

```
interface Node {  
  [...]  
  readonly attribute DOMString      nodeName;  
          attribute DOMString      nodeValue;  
  readonly attribute unsigned short  nodeType;  
  readonly attribute Node            parentNode;  
  readonly attribute NodeList        childNodes;  
  readonly attribute Node            firstChild;  
  readonly attribute Node            lastChild;  
  readonly attribute Node            previousSibling;  
  readonly attribute Node            nextSibling;  
  readonly attribute NamedNodeMap    attributes;  
  readonly attribute Document        ownerDocument;  
  [...]  
};
```

Caratteristiche di ogni nodo

Ogni nodo, come oggetto javascript, possiede proprietà e **metodi**

I metodi:

```
interface Node {  
  [...]  
  Node      insertBefore(in Node newChild,  
                           in Node refChild);  
  Node      replaceChild(in Node newChild,  
                           in Node oldChild);  
  Node      removeChild(in Node oldChild);  
  Node      appendChild(in Node newChild);  
  boolean   hasChildNodes();  
  Node      cloneNode(in boolean deep);  
  [...]  
};
```

Manipolazione del DOM

La cassetta degli attrezzi del DHTML-ista

```
document.getElementById()
```

```
document.getElementsByTagName()
```

```
document.createElement()
```

```
document.createTextNode()
```

```
nuovo_elemento.appendChild()
```

Eventi

La gestione degli eventi, permette di associare del codice (*handlers*) al momento in cui accade "qualcosa"

```
<a href="..."  
onclick="if (confirm('sei sicuro?')) return true; else return false">  
Cancella tutto i tuoi documenti  
</a>
```

Eventi

Elementi diversi di HTML rispondono a eventi diversi

`<body>` → `onload`

`<input>` → `onclick`, `onkeypress`, `onblur`, etc

`<select>` → `onchange`

Eventi

Gli eventi possono essere proprietà (metodi) degli oggetti DOM

```
my_obj = document.getElementById('conferma')
my_obj.onclick = chiedi_conferma

function chiedi_conferma()
{
    return (confirm('Sei sicuro?'))
}
```

In questo modo è però possibile definire UN SOLO handler. Non è possibile aggiungerne in cascata.

Eventi, the W3C way

```
my_obj = getElementById('something')
my_obj.addEventListener('click', my_click_handler, false)
my_obj.addEventListener('focus', my_focus_handler, false)
my_obj.addEventListener('blur', my_blur_handler, false)

my_obj.removeEventListener('click', my_click_handler, false);
```

Internet Explorer?

IE, tuttora, non supporta `addEventListener` e `removeEventListener`.

Lui ha i "suoi" metodi.

```
my_obj = getElementById('something')
my_obj.attachEvent('onclick', my_click_handler)
my_obj.attachEvent('onfocus', my_focus_handler)
my_obj.attachEvent('onblur', my_blur_handler)

my_obj.detachEvent('click', my_click_handler);
```

Event Bubbling



Eventi

Si può cancellare il comportamento di default di un certo evento

DOM → preventDefault

IE → cancelBubble, returnValue

```
evt.cancelBubble = true;
```

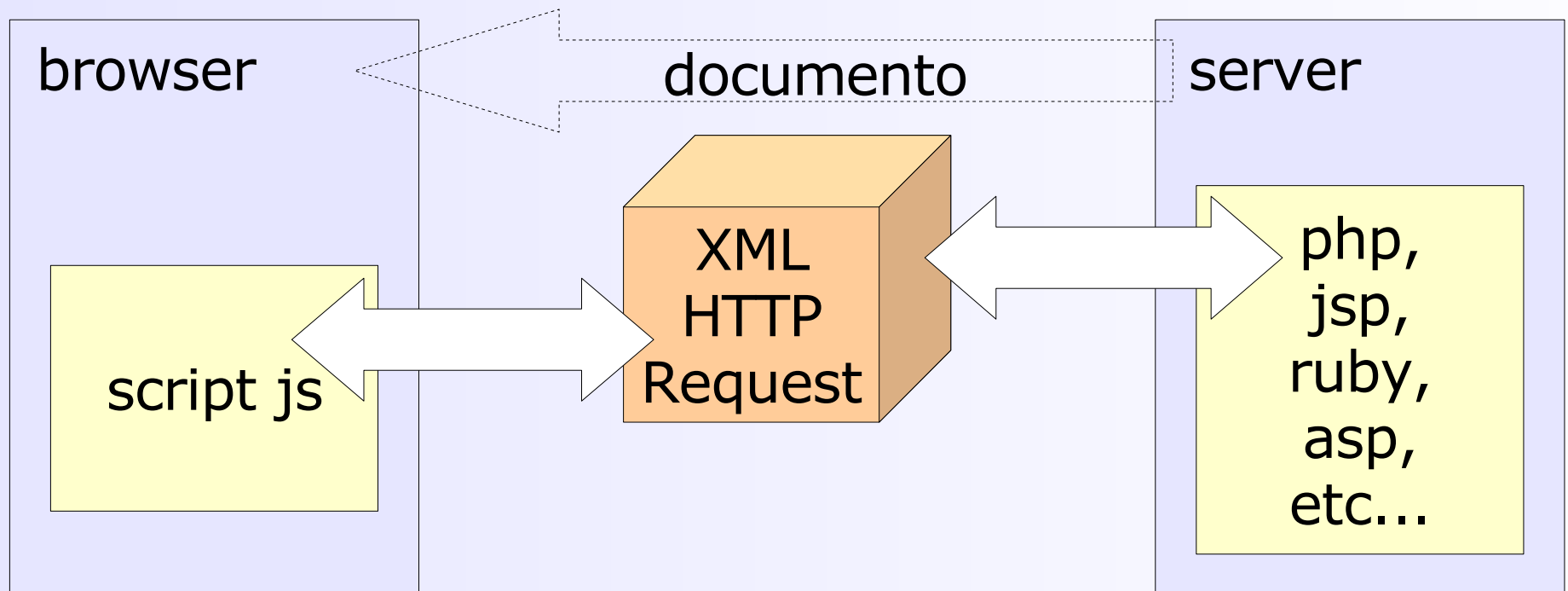
```
if (evt.preventDefault) evt.preventDefault();
```

```
if (evt.returnValue) evt.returnValue = false;
```



Colloquio client/server
L'oggetto **XMLHttpRequest**

L'oggetto "xhr" permette di comunicare via HTTP, dal nostro script con un servizio presente sul server. Sullo **stesso** server.



Genesis

Non è uno **standard ufficiale**.

Creato con Internet Explorer 5 per mezzo di **ActiveX**

Attualmente è supportato dalle ultime versioni dei browser più usati

Una standardizzazione è in corso (WHATWG e DOM Level 3)

Due modalità di funzionamento

Asincrona

Sincrona

Flusso (spedizione)



Flusso (spedizione)

Creazione dell'oggetto
req = new XMLHttpRequest()



Impostazione della funzione *callback*
req.onreadystatechange = processReqChange



Apertura della connessione
req.open("GET", url, true)



Spedizione della richiesta
req.send("Hello!")

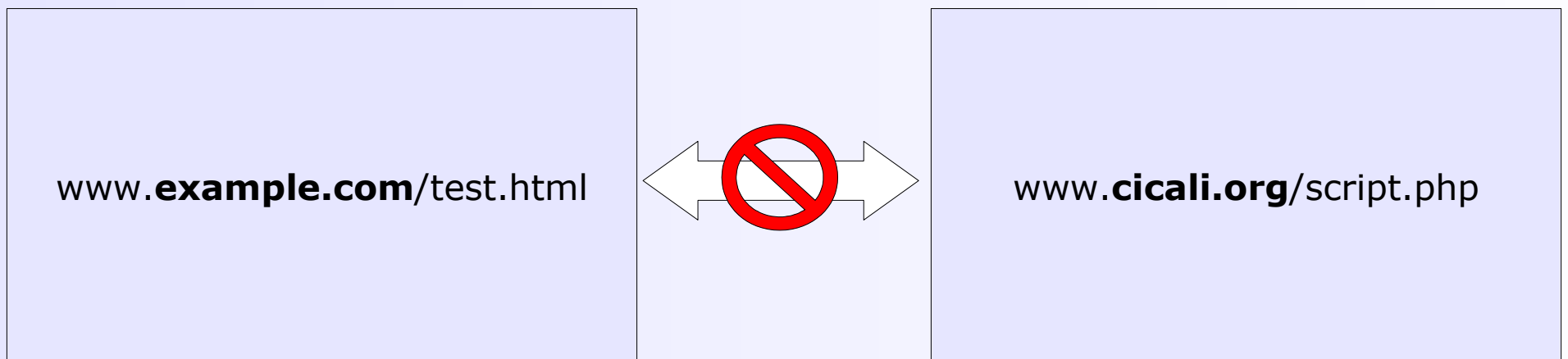
Flusso (ricezione)

La ricezione è gestita dalla funzione *callback* definita

```
function processReqChange ()
{
    /*
    0 = uninitialized
    1 = loading
    2 = loaded
    3 = interactive
    4 = complete
    */
    if (req.readyState == 4)
    {
        if (req.status == 200)
        {
            // La richiesta ha avuto successo... procediamo
        }
        else
        {
            alert("Problemi nella ricezione:\n" + req.statusText);
        }
    }
}
```

Security issues

- Lo *schema* dell'URI può essere solo HTTP
- Il dominio verso il quale effettuare la richiesta deve essere lo stesso dal quale arriva la pagina nel quale è incluso il javascript





A synchronous

J avaScript

A nd

X ML



18 febbraio 2005,

J. J. Garrett scrive il seguente articolo:

Ajax: A New Approach to Web Applications

<http://www.adaptivepath.com/publications/essays/archives/000385.php>

AJAX diventa la *buzzword* del momento

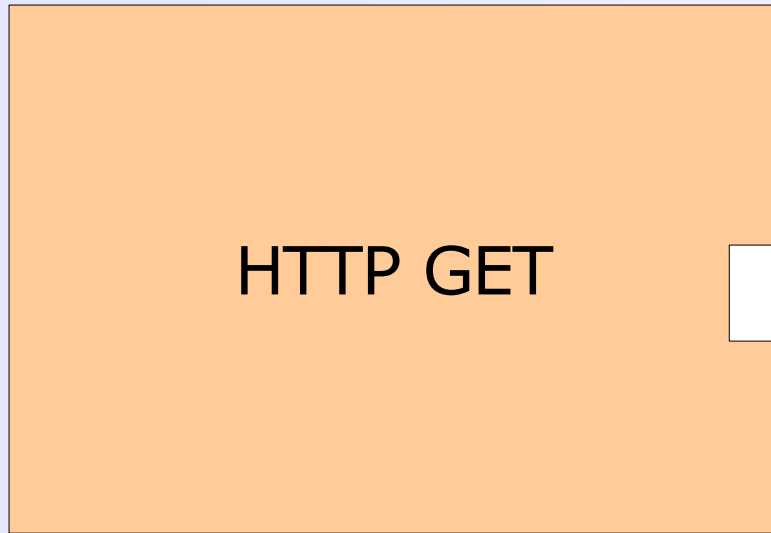
Ajax isn't a technology.
It's really several technologies.

Ajax incorporates:

- standard based presentation using **XHTML** and **CSS**;
- dynamic display and interaction using the **Document Object Model**;
- data interchange and manipulation using **XML** and **XSLT**;
- asynchronous data retrieval using **XMLHttpRequest**;
- and **Javascript** binding everything together.

Xml o JSON?

Javascript



PHP, Java, ASP, Rails, etc..

