Ajax - communication

Andrea Ferracani
andreaferracani@gmail.com

Ajax non era una novità quando Jesse James Garret coniò il nome (asynchronous javascript and xml).

E' un merging di diverse tecnologie e ne eredita tutti i problemi e le limitazioni

E' diventato diffuso grazie ad alcune brilanti applicazioni come Google suggest o Google maps

Ajax è il merging di queste tecnologie:

- Semantic xhtml markup
- Il document object model
- Javascript
- XML

XHTML e DOM

Con ajax funziona qualsiasi pagina html, ma sicuramente un markup semantico con tag e id appropriati rende la manipolazione degli elementi molto più facile

XMLHttpRequest

E' il core di ajax, l'oggetto che serve ad interagire con il server tramite chiamate asincrone. C'è un draft del W3C con le specifiche ma non è implementato da tutti i browser (http://www.w3.org/TR/XMLHttpRequest).
Esiste già da IE 5.0, in cui è un active-x plugin, cioè un componente che poteva essere istanziato così:

```
var request = new ActiveXObject("Microsoft.XMLHTTP");
```

Adesso è stato implementato da tutti (anche explorer 7) in questa forma:

```
var req = new XMLHttpRequest();
```

I metodi tipici di questo oggetto sono:

- open(method, URL[, asynchronous[, userName[, password]]])
- setRequestHeader(label, value), va chiamato dopo open e prima di send
- send(content) trasmette la richiesta, e content opzionale se per es. il metodo è post

- abort(), per fermare la request corrente
- getAllResponseHeaders(), ritorna tutti gli headers come stringa
- getResponseHeader(label), ritorna l'header con una label specifica

FARE UNA REQUEST

```
function stateChangeListener() {
//some code
}
var request = false;
if(window.XMLHttpRequest) {
var request = new window.XMLHttpRequest();
} else if (window.ActiveXObject) {
var request = new window.ActiveXObject('Microsoft.
XMLHTTP');
}
```

```
if(request) {
request.onreadystatechange = stateChangeListener;
request.open('GET', '/your/script/?
var=value&var2=value', true);
request.send(null);
}
```

Nel caso si usi il metodo post la request sarebbe:

```
request.open('POST', '/your/script/', true);
request.send('var=value&var2=value');
```

!! Javascript non può accedere a file system o da file input, limitazione dei browser

Il metodo onreadystatechange, per l'esattezza un gestore d'evento, sarà chiamato diverse volte durante la request registrando lo stato di una certa proprietà dell'oggetto.

Le proprietà di XMLHttpRequest sono:

- readyState è un intero che regista gli stati della request:
  - 0 uninitialized
  - 1 loading
  - 2 loaded
  - 3 interactive
  - 4 complete

- responseText se il server risponde una stringa
- responseXML un document object che rappresenta l'XML di risposta se c'è l'header giusto nel documento
- status, lo stato della richiesta appunto 404 not found oppure 200 ok per es. Vedi (http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html)
- statusText
- onreadystatechange che deve contenere le funzioni da eseguire relative allo stato della richiesta

es.

```
function stateChangeListener() {
// Switch functionality depending on the state of
the request
switch (request.readyState) {
case 1:
// Loading
break;
case 2:
// Loaded
break;
```

```
case 3:
// Interactive
break;
case 4:
// Complete
if (request.status == 200) {
// Do something with request.responseText
// or request.responseXML
} else {
// There was a possible error code in request.status
// with a message as reported in request.statusText
}break;}}
```

L'header della risposta deve avere un Content-Type di tipo application/xml perchè XMLHttpRequest capisca che quello è xml altrimente assumerà che sia testo.

Funzione da eseguire quando la richiesta è stat completata

```
// Create the method you want to run.
function alertAndDoWhatever(r) {
// r is the request object passed in
// through the onreadystatechange listener
alert(r.responseText);
```

```
}
request.onreadystatechange = function() {
// Run the method when the request is complete.
if(request.readyState == 4 && request.status ==
'200') {
// The request completed successfully so call your
method.
alertAndDoWhatever(request);
}
}
// Open the request.
request.open('GET', '/yourscript/?var=value', true);
// Send it.
request.send(null);
```

Si possono inviare header personalizzati in una richiesta. Un header tipico:

Host=advanceddomscripting.com
User-Agent=Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; å
en-US; rv:1.8.1.1)
Gecko/20061204 Firefox/2.0.0.1
Accept=text/xml,application/xml,
application/xhtml+xml,text/html;q=0.9,å
text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language=en-us,en;q=0.5
Accept-Encoding=gzip,deflate
Accept-Charset=ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive=300

Connection=keep-alive
Referer=http://ww.google.com/example.html

es.

```
request.open('GET', '/yourscript/?var=value', true);
request.setRequestHeader
('My-Special-Header','AjaxRequest');
request.setRequestHeader('Sent-By','Jeff');
request.send();
```

Per ritrovarlo lato server in PHP si usa un array globale:

```
<?php
if (isset($_SERVER['HTTP_MY_SPECIAL_HEADER'])) {
// Respond to the XMLHttpRequest
} else {
// Respond to the traditional request
}
?>
```

attenzione agli hyphen ed al prefisso http!

lato client invece:

```
switch(request.getResponseHeader('Content-Type'))
{
case 'text/javascript':
// use request.responseText as JavaScript;
break;
case 'text/xml':
case 'application/xml':
// use request.responseXML or read request.
responseText as XML
break;
case 'text/html':
// use request.responseText as HTML;
break;}
```

Si può usare Firebug per controllare gli header

XML

responseXML è la parte più importante della nostra comunicazione, possiamo parsare l'xml usando i metodi del DOM:

```
var messages = request.responseXML.
getElementsByTagName('messages');
for(var i=0 ; i < messages.length ; i++) {
Mine.$('example').appendChild(messages[i]);
}
```

Inoltre possiamo associare direttamente un file xslt al nostro xml per produrre html:

```
var xsltProcessor = new XSLTProcessor();
var xslStylesheet;
var xmlDoc;
// Retrieve an XSL file asynchronously.
var requestXsl = new XMLHttpRequest();
requestXsl.onreadystatechange = function(request)
{
xslStylesheet = request.responseXML;
}
requestXsl.open("GET", "example1.xsl", true);
```

```
requestXsl.send(null);
// Retrieve an XML file asynchronously.
var requestXml = new XMLHttpRequest();
requestXml.onreadystatechange = function(request)
{
xmlDoc = request.responseXML;
}
requestXml.open("GET", "example1.xsl", true);
requestXml.send(null);
var processor = function() {
if(xslStylesheet && xmlDoc) {
clearInterval(this);
```

```
// Transform the XML using the XSLT.
xsltProcessor.importStylesheet(xslStylesheet);
var fragment = xsltProcessor.transformToFragment(
xmlDoc, document
);
ADS.$('example').appendChild(fragment);
}
}
// Check every 200 milliseconds to see if the files
are loaded.
setInterval(processor,200);
```

PLAIN TEXT

Per richieste semplici si può rispondere con una stringa:

```
request.onreadystatechange = function() {
if(this.readyState == 4 && request.status == 200) {
if(request.responseText == 't') {
// The server processing worked
} else {
// The server processing failed
}
}}
```

HTML

oppure con dell'html:
The response was successful

```
request.onreadystatechange = function() {
if(request.readyState == 4 && request.status ==
200) {
Mine.$('example').innerHTML = request.
responseText;
}
}
```

- ma non è riutilizabile
- IE ha problemi con la proprietà innerHTML se nel markup ci sono table o select list

JAVASCRIPT

potete anche restituire javascript code come

```
alert('The response was successful.');
```

Si usa la funzione eval() sulla proprietà responseText per eseguire il codice:

```
request.onreadystatechange = function() {
if(request.readyState == 4 && request.status ==
200) {
eval(request.responseText);
}
}
```

ma non è una buona soluzione per ragioni di sicurezza e per la separazione delle varie funzionalità

JSON
si può anche restituire un oggetto:

```
{
message : 'The response was successful',
type : 'success'
}
```

anche in questo caso si deve usare eval();

es.

```
request.onreadystatechange = function() {
if(this.readyState == 4 && request.status == 200) {
// Evaluate the JSON to populate the response
object.
var response = eval('(' + this.responseText + ')');
// Do something with response.
// With innerHTML
ADS.$('example').innerHTML = '<p class="'
+ response.type
+ '">'
+ response.message
+ </p>;
```

```
// or alert
alert(response.message);
// or DOM using methods...
var p = document.createElement('P')
p.className = response.type;
p.appendChild(document.
createTextNode(response.message));
ADS.$('example').appendchild(p);
}
}
```

anche questo metodo ha problemi di vulnerabilità, bisognerebbe usare un parser json che controlli

la conformità dell'oggetto:

http://www.json.org/json.js

UN OGGETTO RIUSABILE

Queste funzioni di comunicazione possono essere incapsulate neli metodi di un oggetto:

- getRequestObject();
- ajaxRequest

```
/* Set up the various parts of an XMLHttpRequest
Object */
function getRequestObject(url,options) {
// Initialize the request object.
var req = false;
if(window.XMLHttpRequest) {
var req = new window.XMLHttpRequest();
} else if (window.ActiveXObject) {
var req = new window.ActiveXObject('Microsoft.
XMLHTTP');
}
```

etc. vedi ajax_communication/Mine.js

```
/* Send an XMLHttpRequest using a quick wrapper
around the
getRequestObject and the send method. */

function ajaxRequest(url,options) {
var req = getRequestObject(url,options);
return req.send(options.send);
}
window['Mine']['ajaxRequest'] = ajaxRequest;
```

- method: il metodo della request, default GET
- send: stringa pzionale da inviare, default null
- loadListener: funzione da eseg. quando readyState = 1 (loading)
- loadedListener: readyState = 2
- interactiveListener: readyState = 3
- jsResponseListener: quando il server restituisce js come stringa - eval
- jsonResponseListener: si restituisce json, content-type: application/json - eval
- xmlResponseListener: si restituisce xml - application/xml, application/xhtml+html
- htmlResponseListener, text/html

- completeListener: viene incocato sempre se nessun case del precedente switch viene eseguito
- errorListener: eseguito nel caso response status != 200

Usage:

```
Mine.ajaxRequest('/path/to/script/',{
method:'GET',
completeListener:function() {
alert(this.responseText);
}
});
```

DOMANDE DA PORSI PRIMA DI USARE AJAX

- limita in qualche modo l'utente dall'accesso alle informazioni?
- spezza la consistenza dell'interazione?
- aumenta i costi e i tempi di sviluppo al fine di creare elementi non necessari?
- aumenta la complessità d'uso?
- vogliamo solo che la nostra interfaccia sia cool?

Se rispondiamo si a una di queste domande allora è il caso di non usare ajax

Il contenuto primario della vostra applicazione o sito non deve dipendere da ajax

Inoltre bisogna tener conto degli utenti con disabilità che usano browser che non suportano applicazioni mouse based

http://www.w3.org/WAI/References/Browsing

Il primo problema che si pone è il security domain implementato da tutti i browser: potete fare request solo all'interno del vostro dominio, non richiedere informazioni da altri

Vedi XssHttpRequest object in

ajax_communication/Mine.js

è un oggetto che funziona in modo simile a xmlHttpRequestObject ma genera uno script nell'header della pagina che attraverso la proprietà src può chiamare un file che sta in un altro dominio, la cosa importante è che il file sul server deve restituire un oggetto JSON:

```
<?php
header('Content-type: text/javascript');
// Only allow number and letters and underscores
in the callback.
$callback = preg_replace(
'/[^A-Z0-9_]/i',
''
,
$_GET['XSS_HTTP_REQUEST_CALLBACK']
);
```

quando si crea l'oggetto si appende anche all'url un parametro get che setta la callback da eseguire quando si fa il load dello script

```
echo "/* XSS request for callback: $callback */\n";
if($callback) {

echo
<<<JSON
{$callback}({
message:'response ok'
});
JSON;
}
?>
```

dunque rstituiamo un oggetto json passandogli un altro oggetto come parametro con una proprietà di risposta

USO:

```
Mine.xssRequest(
'http://path/xssRequest/responder.php',{
completeListener:function() {
    alert(this.responseJSON.message);
},
errorListener:function() {
    alert(this.statusText);
}});
```

BACK BUTTONS and BOOKMARKS

Il pulsante indietro in una pagina che usa ajax non riporta alla chiamata precedente ma al sito precedente

Così per un bookmark, si rimanda sempre alla prima pagina se la navigazione è fatta in ajax

Si possono però per esempio usare degli hash per certe immagini http://localhost/ /browser/#photo/1 che possono essere messe nei bookmark o utilizzati per tornare indietro

Dovremo dunque creare un oggetto che individui i cambiamenti della barra del browser e consenta di registrare degli eventi per questo ed implementare i vari metodi usati da browser

- firefox e opera si comportano bene
- IE non ignora gli hash dunque dovremo usare un iframe ed inserire l'hash in un parametro get
- Safari ha dei problemo con window.location.href, pur segando i vari hash in history ed history. length

vedi ajax_communication/Mine - actionPager method

CHIAMATE ASINCRONE

Avvengono tutte nello stesso momento, ma non c'è modo di prevedere il tempo di risposta:

- request
- network di computer
- server
- response
- network di computer
- pc

Vedi Mine/latency e localhost/latency

pensate ai problemi che possono nascere se sto scrivendo su db, per esempio in un carrello di acquisti o nello spostamento di blocchi.

Una soluzione è settare false nel parametro asynchronous del metodo open()... ma causa il freezing del browser

oppure una queue che consente di continuare il proprio lavoro mentre le richiese vengono effettuate una dopo l'altra:

metodo ajaxRequestQueue() :

USO :

```
// Request 1 in queue 1
Mine.ajaxRequestQueue('/your/script/',{
completeListener: funcion() {
alert(this.responseText);
}
},'Queue1');
// Request 2 in queue 2
Mine.ajaxRequestQueue('/your/script/',{
completeListener: funcion() {
alert(this.responseText);
}
},'Queue2');
```

```
// Request 3 in queue 1 will wait until
// request 1 is done.

Mine.ajaxRequestQueue('/your/script/',{
completeListener: funcion() {
alert(this.responseText);
}
},'Queue1');
```

Vedi ajaxRequestQueue
e
localhost/master_course/ajaxRequestQueue

Bisogna inoltre considerare il fatto che un utente tende a premere più volte su un bottone, dunque è opportuno disabilitarlo mentre viene effettuata una richiesta:

```
<input type="submit" id="buttonID">

Mine.ajaxRequest('/your/script/',{
loadListener:function() {
// Disable the button while loading
Mine.$('buttonID').disabled = 'disabled';
},
```

```
completeListener: funcion() {
// Enabled the button when successful
Mine.$('buttonID').disabled = '';
alert(this.responseText);
},
errorListener: function() {
// Enabled the button after an error as well
Mine.$('buttonID').disabled = '';
alert('Oops, please try again: ' + this.statusText);
}
});
```

ovviamente è una soluzione che ne drag and drop non funziona!!

UPLOAD IN Ajax

Teoricamente xmlHttpRequest non avrebbe alcuna limitazione ad uploadare qualsiasi tipo di dato il fatto è che il browser non può accedere ai file per ragioni di sicurezza.

Se si potessero leggere i dati binari si potrebbe fare una cosa tipo questa:

```
var request = false;
if(XMLHttpRequest) {
    var request = new XMLHttpRequest();
} else if (ActiveXObject) {
    var request = new
    ActiveXObject('Microsoft.XMLHTTP');
}

if(request) {
var boundary = '--ExampleBoundaryString';
var requestBody =
boundary + '\n'
+ 'Content-Disposition: form-data; name="'
exampleText"'\n\n'
```

```
+ exampleText + '\n\n'
+ boundary + '\n'
+ 'Content-Disposition: form-data; name="myfile";
filename=
"example.zip"\n'
+ 'Content-Type: application/octet-stream\n\n'
+ escape(BinaryContent) + '\n'
+ boundary;
request.open('POST', url, true);
request.setRequestHeader('Content-Type',
'multipart/form-data; boundary="'
+ boundaryString
+ '"');
request.setRequestHeader('Connection', 'close');
```

```
request.setRequestHeader('Content-Length',
requestBody.length);
request.send(requestBody);
}
```

oppure firefox consente di caricare immagini editando alcuni settings di about:config; ma funzionerebbe solo in alcuni browser

Una soluzione è usare il vecchio POST con un iframe come target se non si vuole ricaricare la pagina:

```
<h1>Form with an iframe as the target</h1>
<form action="script/" target="formTarget">
<div>
<label for="example">Some text</label>
<input type="text" name="example" id="example"
value="" />
</div>
<div>
<input type="submit" value="Send it to the iframe"
/>
</div>
</form>
<iframe name="formTarget" id="formTarget"
></iframe>
```

PROGRESS INDICATORS