EBook Gratuito

APPRENDIMENTO

ajax

Free unaffiliated eBook created from
Stack Overflow contributors.

#ajax

Sommario

Di

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: ajax

It is an unofficial and free ajax ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official ajax.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capitolo 1: Iniziare con ajax

Osservazioni

AJAX **(sincrono J** avascript **un** nd **X** ML) consente di richiedere dati esterni **senza** bloccare l'esecuzione del codice. In molti casi questo è implementato nel richiedere pezzi di una pagina o informazioni da un server (tramite XMLhttpRequests) e quindi elaborarli e visualizzarli usando javascript.

La natura non bloccante di AJAX è ciò che lo rende un modello software molto diffuso. Poiché javascript sta bloccando nel browser, una chiamata esterna sincrona renderebbe il browser non rispondente per la durata della chiamata fino a quando non ha restituito dati o scaduto. In effetti, la tua applicazione dipende interamente dall'architettura esterna e dalla sua efficacia.

Le chiamate AJAX sono solitamente astratte per fornire funzionalità aggiuntive o leggibilità, ma le implementazioni sono (di solito) costruite sulla specifica XMLHttpRequest .

Examples

Installazione o configurazione

Cos'è AJAX?

> AJAX sta per Asynchronous JavaScript e XML. In breve, è l'uso dell'oggetto XMLHttpRequest per comunicare con gli script sul lato server. Può inviare e ricevere informazioni in una varietà di formati, inclusi JSON, XML, HTML e persino file di testo. - Mozilla Developer Network 2016

Il modo più semplice per implementare AJAX, specialmente se stai pianificando di comunicare con i server, è usare jQuery.

Cos'è jQuery?

> jQuery è una libreria JavaScript veloce, piccola e ricca di funzionalità. Rende le cose come attraversamento di documenti HTML e manipolazione, gestione degli eventi, animazione e Ajax molto più semplice con un'API facile da usare che funziona su una moltitudine di browser. -jquery.com

Per coloro che non hanno usato molto jQuery, pensatelo come funzioni che possiamo usare per rendere più facile la nostra vita. Questo è perfetto per l'utilizzo con AJAX in quanto riduce la quantità di codice che dobbiamo scrivere per realizzare la stessa cosa!

Come aggiungere jQuery al tuo sito web

Se hai bisogno di usare Ajax, devi aggiungere jQuery al tuo progetto. http://jquery.com/download/ In questo link puoi vedere molti modi per aggiungere jquery. Puoi usare la versione scaricata di jQuery o puoi usare un CDN. http://jquery.com/download/#jquery-39-s-cdn-provided-by-maxcdn . Ma c'è qualche rischio per la sicurezza se usi la CDN. Becauese il progetto chiama ad usare jquery, quindi un hacker potrebbe manipolare la chiamata. Quindi meglio se potessi usare la versione scaricata. Vediamo come aggiungere jquery al progetto html. È facile. Il primo esempio è utilizzare la fonte scaricata. Usa questo link per scaricare http://jquery.com/download/#jquery . Se vuoi utilizzare jquery, ti suggerisco di scaricare **Scarica il file compresso, produzione jQuery 3.1.1** Quando lo scarichi aggiungi jquery-version.min.js al posto appropriato (come la cartella javascript del tuo progetto) Quindi aggiungi il tag con src = jquery / posizione come sotto.

```
<head>

<script src="path/from/html/page/to/jquery.min.js"></script>

</head>
```

Vediamo come usare un CDN. Questo link http://jquery.com/download/#using-jquery-with-a-cdn puoi vedere vari CDN (Content Delivery Network).

```
<head>

<script   src="https://code.jquery.com/jquery-3.1.1.min.js"   integrity="sha256-
hVVnYaiADRTO2PzUGmuLJr8BLUSjGIZsDYGmIJLv2b8="   crossorigin="anonymous"></script>

</head>
```

Come puoi vedere, devi solo aggiungere i tag che il provider CDN fornisce al. Ora aggiungi alcuni script alla pagina html per verificare che funzioni.

```
<script>
$(document).ready(function(){
    alert("jQuery Works")
});
</script>
```

Se vedi **jQuery funziona come** avviso, significa che lo hai aggiunto correttamente.

Semplice esempio jQuery per comunicare con il server

Tratto dal sito Web dell'API `jQuery.ajax` :

```
$.ajax({
    method: "POST",
    url: "some.php",
    data: {
        name: "John",
        location: "Boston"
```

```
    },
    success: function(msg) {
        alert("Data Saved: " + msg);
    },
    error: function(e) {
        alert("Error: " + e);
    }
});
```

Questa parte di codice, dovuta a jQuery, è facile da leggere e capire cosa sta succedendo.

- `$.ajax` - questo bit chiama di jQuery `ajax` funzionalità.
- `method: "POST"` - questa riga qui dichiara che useremo un metodo POST per comunicare con il server. Leggi su tipi di richieste!
- `url` - questa variabile dichiara dove la richiesta sta per essere **inviato** a. Si sta inviando una richiesta **di** qualche parte. Questa è l'idea.
- `data` - piuttosto semplice. Questi sono i dati che invii con la tua richiesta.
- `success` - questa funzione qui scrivi per decidere cosa fare con i dati che ricevi `msg` ! come suggerisce l'esempio, al momento sta solo creando un avviso con il `msg` che viene restituito.
- `error` : questa funzione viene scritta per visualizzare i messaggi di errore o per fornire le azioni da eseguire quando la richiesta `ajax` subito errori.
- In alternativa alla `.done` è

  ```
  success: function(result) {
      // do something
  });
  ```

Sincronizzazione - richieste asincrone asincrone

Chiamata ajax asincrona

Con questo tipo di chiamata ajax, il codice non attende il completamento della chiamata.

```
$('form.ajaxSubmit').on('submit',function(){
    // initilization...
    var form   = $(this);
    var formUrl    = form.attr('action');
    var formType   = form.attr('method');
    var formData   = form.serialize();

    $.ajax({
        url: formUrl,
        type: formType,
        data: formData,
        async: true,
        success: function(data) {
            // .....
        }
```

```
    });
    //// code flows through without waiting for the call to complete
    return false;
});
```

Chiamata sincrona ajax

Con questo tipo di chiamata ajax, il codice attende la chiamata da completare.

```
$('form.ajaxSubmit').on('submit',function(){
    // initilization...
    var form   = $(this);
    var formUrl    = form.attr('action');
    var formType   = form.attr('method');
    var formData   = form.serialize();
    var data = $.ajax({
            url: formUrl,
            type: formType,
            data: formData,
            async: false
        }).responseText;
    //// waits for call to complete
    return false;
});
```

Richiesta Ajax semplice inviata utilizzando l'oggetto XMLHttpRequest

```
var httpRequest = new XMLHttpRequest();

httpRequest.onreadystatechange = getData;

httpRequest.open('GET', 'https://url/to/some.file', true);
httpRequest.send();

function getData(){
    if (httpRequest.readyState === XMLHttpRequest.DONE) {
        alert(httpRequest.responseText);
    }
}
```

`new XMLHttpRequest()` crea un nuovo oggetto *XMLHttpRequest* - questo è ciò con cui invieremo la nostra richiesta

Il bit `onreadystatechange` indica alla nostra richiesta di chiamare `getData()` ogni volta che cambia lo stato

`.open()` crea la nostra richiesta - questo accetta un *metodo di richiesta* (' **GET** ', ' **POST** ', ecc.), un url della pagina che stai interrogando, e opzionalmente, se la richiesta debba essere **asincrona** o meno

`.send()` invia la nostra richiesta - accetta facoltativamente i dati da inviare al server come `.send(data)`

infine, `getData()` è la funzione che abbiamo detto dovrebbe essere chiamata ogni volta che

cambia lo stato della richiesta. se readyState è uguale a **DONE** allora avvisa il `responseText` che è solo i dati ricevuti dal server.

Maggiori informazioni possono essere trovate nella guida introduttiva su MDN.

Utilizzando ajax in vanilla javascript con una semplice richiamata

Ecco la nostra funzione per creare una semplice chiamata ajax scritta in vanilla javascript (non es2015):

```
function ajax(url, callback) {
    var xhr;

    if(typeof XMLHttpRequest !== 'undefined') xhr = new XMLHttpRequest();
    else {
        var versions = ["MSXML2.XmlHttp.5.0",
                        "MSXML2.XmlHttp.4.0",
                        "MSXML2.XmlHttp.3.0",
                        "MSXML2.XmlHttp.2.0",
                        "Microsoft.XmlHttp"]

         for(var i = 0, len = versions.length; i < len; i++) {
            try {
                xhr = new ActiveXObject(versions[i]);
                break;
            }
            catch(e){}
         } // end for
    }

    xhr.onreadystatechange = ensureReadiness;

    function ensureReadiness() {
        if(xhr.readyState < 4) {
            return;
        }

        if(xhr.status !== 200) {
            return;
        }

        // all is well
        if(xhr.readyState === 4) {
            callback(xhr);
        }
    }

    xhr.open('GET', url, true);
    xhr.send('');
}
```

e potrebbe essere usato come:

```
ajax('myFile.html', function(response) {
    document.getElementById('container').innerHTML = response.responseText;
});
```

Se si desidera utilizzare Ecmascript 6 (noto anche come es2015), è possibile utilizzare il metodo **fetch** , che restituisce una promessa:

```
fetch('myFile.json').then(function(res){
    return res.json();
 });
```

Per ulteriori informazioni su es2015 Promises segui il link seguente: Promises

Esecuzione di una chiamata AJAX asincrona mediante TypeScript

Aggiungere un prodotto a un carrello

Nell'esempio seguente viene illustrato come aggiungere un prodotto (o qualcosa) a una tabella di database in modo asincrono, utilizzando AJAX e TypeScript.

```
declare var document;
declare var xhr: XMLHttpRequest;

window.onLoad = () =>
{
    Start();
};

function Start()
{
    // Setup XMLHttpRequest (xhr).
    if(XMLHttpRequest)
    {
        xhr = new XMLHttpRequest();
    }
    else
    {
        xhr = new ActiveXObject("Microsoft.XMLHTTP");
    }

    AttachEventListener(document.body, "click", HandleCheckBoxStateChange);
}

function AttachEventListener(element: any, e, f)
{
    // W3C Event Model.
    if(element.addEventListener)
    {
        element.addEventListener(e, f, false);
    }
    else if(element.attachEvent)
    {
        element.attachEvent("on" + e, (function(element, f)
        {
            return function()
            {
                f.call(element, window.event);
            };
        })
```

```
        (element, f));
    }

    element = null;
}

function HandleCheckBoxStateChange(e)
{
    var element = e.target || e.srcElement;

    if(element && element.type == "checkbox")
    {
        if(element.checked)
        {
            AddProductToCart(element);
        }
        else
        {
            // It is un-checked.
            // Remove item from cart.
        }
    }
    else
    {
        break;
    }
}

AddProductToCart(e)
{
    var element = <HTMLInputElement>document.getElementById(e.id);

    // Add the product to the Cart (Database table)
    xhr.onreadystatechange = function()
    {
        if(xhr.readyState == 4)
        {
            if(xhr.status == 200)
            {
                if(element != null)
                {
                    console.log("200: OK");
                }
                else
                {
                    console.log(":-(");
                }
            }
            else
            {
                // The server responded with a different response code; handle accordingly.
                // Probably not the most informative output.
                console.log(":-(");
            }
        }
    }

    var parameters = "ProductID=" + encodeURIComponent(e.id) + "&" + "Action=Add&Quantity=" +
element.value;

    xhr.open("POST", "../Cart.cshtml");
```

```
    xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    xhr.setRequestHeader("Content-length", parameters.length.toString());
    xhr.setRequestHeader("Connection", "close");

    xhr.send(parameters);

    return e.id;
}
```

Per completare questo esempio, aggiorna il codice sul lato server per inserire effettivamente questi dati nel database. Il codice sopra presuppone che stai usando C # e hai un file `Cart.cshtml` . Tuttavia, è sufficiente sostituire `cshtml` con `php` e scrivere la propria logica lato server utilizzando la lingua desiderata.

Leggi Iniziare con ajax online: https://riptutorial.com/it/ajax/topic/1082/iniziare-con-ajax

Capitolo 2: callback

Examples

Interpretazione degli errori con il callback "errore"

Gli errori, se gestiti correttamente dal server, verranno restituiti al client con uno specifico codice di stato HTTP diverso da 2xx (vedere RFC 2616 sezione 10).

Si consiglia di catturare globalmente i propri errori dal proprio `$.ajaxSetup()` come mostrato nell'esempio sotto. Pertanto, tutti gli errori provenienti dalle chiamate ajax verranno automaticamente interpretati dall'impostazione Ajax.

```
$.ajaxSetup({
    error: function (jqXHR, exception, errorThrown) {
        var message;
        var statusErrorMap = {
            '400': "Server understood the request, but request content was invalid.",
            '401': "Unauthorized access.",
            '403': "Forbidden resource can't be accessed.",
            '500': "Internal server error.",
            '503': "Service unavailable."
        };
        if (jqXHR.status) {
            message = statusErrorMap[jqXHR.status];
            if (!message) {
                message = "Unknown Error.";
            }
        } else if (exception == 'parsererror') {
            message = "Error.\nParsing JSON Request failed.";
        } else if (exception == 'timeout') {
            message = "Request Time out.";
        } else if (exception == 'abort') {
            message = "Request was aborted by the server";
        } else {
            message = "Unknown Error.";
        }

        // How you will display your error message...
        console.log(message);
        console.log(errorThrown);
    }
});
```

Si consiglia inoltre di "sovraccaricare" la richiamata di `error` in uno specifico `$.ajax()` quando si attende un messaggio di errore specifico.

```
$.ajax({
    url: './api',
    data: { parametersObject },
    type:'post',
    dataType: 'json',
    success:function(output){
```

```
        // Interpret success
    },
    error: function(xhr,textStatus,ErrorThrown){
        // Specific error will not be interpreted by $.ajaxSetup
    }
});
```

Leggi callback online: https://riptutorial.com/it/ajax/topic/7175/callback

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con ajax	acupajoe, adielhercules, Alessio Cantarella, Community, delete me, JhWebDevGuy, jignesh prajapati, MAZux, Menuka Ishan, Nicholas Qiao, TimTheEnchanter, Tolga Evcimen
2	callback	maxime_039