EBook Gratis

APRENDIZAJE

ajax

Free unaffiliated eBook created from
Stack Overflow contributors.

#ajax

Tabla de contenido

Acerca de

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

Capítulo 1: Empezando con ajax

Observaciones

AJAX (**un J** avaScript síncrono **y** nd **X** ML) le permite solicitar datos externos **sin** bloquear la ejecución del código. En muchos casos, esto se implementa al solicitar partes de una página o información de un servidor (a través de XMLhttpRequests) y luego procesarla y mostrarla usando javascript.

La naturaleza de no bloqueo de AJAX es lo que lo convierte en un patrón de software tan extendido. Debido a que javascript está bloqueando en el navegador, una llamada externa sincrónica haría que el navegador no responda durante la duración de la llamada hasta que devuelva los datos o se agote el tiempo de espera. De hecho, su aplicación depende totalmente de la arquitectura externa y del rendimiento de la misma.

Las llamadas AJAX generalmente se abstraen para proporcionar funcionalidad adicional o legibilidad, pero las implementaciones (generalmente) se basan en la especificación XMLHttpRequest .

Examples

Instalación o configuración

¿Qué es AJAX?

> AJAX significa JavaScript asíncrono y XML. En pocas palabras, es el uso del objeto XMLHttpRequest para comunicarse con scripts del lado del servidor. Puede enviar y recibir información en una variedad de formatos, incluidos JSON, XML, HTML e incluso archivos de texto. -Mozilla Developer Network 2016

La forma más sencilla de implementar AJAX, especialmente si está planeando comunicarse con los servidores, es utilizando jQuery.

¿Qué es jQuery?

> jQuery es una biblioteca de JavaScript rápida, pequeña y con muchas funciones. Hace que cosas como la manipulación y manipulación de documentos HTML, el manejo de eventos, la animación y Ajax sean mucho más simples con una API fácil de usar que funciona en una multitud de navegadores. -jquery.com

Para aquellos que no han usado mucho jQuery, piense en ello como funciones que podemos usar para hacer nuestras vidas más fáciles. ¡Esto es perfecto para usar con AJAX ya que reduce la cantidad de código que tenemos que escribir para lograr lo mismo!

Cómo agregar jQuery a tu sitio web

Si necesita utilizar Ajax, debe agregar jQuery a su proyecto. http://jquery.com/download/ En este enlace puede ver muchas formas de agregar jquery. Puede usar la versión descargada de jQuery o puede usar un CDN. http://jquery.com/download/#jquery-39-s-cdn-provided-by-maxcdn . Pero hay algún riesgo de seguridad si usas CDN. Porque el proyecto llama a usar jquery, por lo que un pirata informático podría manipular la llamada. Así que mejor si pudieras usar la versión descargada. Vamos a ver cómo agregar jquery al proyecto html. Es fácil. El primer ejemplo es usar la fuente descargada. Utilice este enlace para http://jquery.com/download/#jquery descargar. Si solo desea utilizar jquery, le sugiero que descargue **Descargue jQuery 3.1.1 comprimido y de producción** Cuando lo descargue, agregue jquery-version.min.js al lugar apropiado (como la carpeta javascript de su proyecto) Luego simplemente agregue la etiqueta con src = jquery / location como abajo.

```
<head>

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

</head>
```

Vamos a ver cómo usar un CDN. En este enlace http://jquery.com/download/#using-jquery-with-a-cdn puede ver varios CDN (Red de entrega de contenido).

```
<head>

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

</head>
```

Como puede ver, solo tiene que agregar las etiquetas que el proveedor de CDN suministra a. Ahora agregue algunos scripts a la página html para verificar que esté funcionando.

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

Si ve que la alerta de **jQuery funciona** , eso significa que la agregó correctamente.

Ejemplo simple de jQuery para comunicarse con el servidor

Tomado del sitio web de la 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);
    }
});
```

Este fragmento de código, debido a jQuery, es fácil de leer y entender lo que está pasando.

- `$.ajax` - este bit llama a la funcionalidad `ajax` de jQuery.
- `method: "POST"` : esta línea aquí declara que vamos a utilizar un método POST para comunicarnos con el servidor. Lea sobre tipos de solicitudes!
- `url` - esta variable se declara en el que se va a **enviar** a la solicitud. Estás enviando una solicitud **a** alguna parte. Esa es la idea.
- `data` - bastante sencillo Estos son los datos que está enviando con su solicitud.
- `success` - esta función aquí se escribe para decidir qué hacer con los datos que vuelvas `msg` ! como sugiere el ejemplo, actualmente solo está creando una alerta con el `msg` que se devuelve.
- `error` : esta función aquí se escribe para mostrar mensajes de error o para proporcionar acciones que funcionen cuando la solicitud `ajax` pasó por errores.
- una alternativa a `.done` es

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

Sync - Async peticiones ajax

Llamada asíncrona ajax

Con este tipo de llamada ajax, el código no espera a que se complete la llamada.

```
$('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;
});
```

Llamada ajax síncrona

Con este tipo de llamada ajax, el código espera a que se complete la llamada.

```
$('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;
});
```

Solicitud simple de Ajax enviada utilizando el objeto 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 nuevo objeto *XMLHttpRequest* . Esto es lo que enviaremos a nuestra solicitud.

El bit `onreadystatechange` indica a nuestra solicitud que llamemos a `getData()` cada vez que cambie su estado.

`.open()` crea nuestra solicitud: toma un *método de solicitud* (' **GET** ', ' **POST** ', etc.), una URL de la página que está consultando y, opcionalmente, si la solicitud debe ser **asíncrona.**

`.send()` envía nuestra solicitud - esto opcionalmente acepta datos para enviar al servidor como `.send(data)`

Finalmente, `getData()` es la función que hemos dicho que debe llamarse cada vez que **cambie el estado de** nuestra solicitud. Si el readyState es igual a **DONE,** entonces alerta el texto de `responseText` que es solo los datos recibidos del servidor.

Se puede encontrar más información en la guía de inicio en MDN.

Usando ajax en javascript de vainilla con una simple devolución de llamada

Esta es nuestra función para crear una llamada ajax simple escrita en vanilla javascript (no 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('');
}
```

y podría ser utilizado como:

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

Si desea usar Ecmascript 6 (también conocido como es2015) puede usar el método de **búsqueda** , que devuelve una promesa:

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

Para leer más sobre Promesas es2015, siga el enlace a continuación: Promesas

Realización de una llamada AJAX asíncrona utilizando TypeScript

Agregar un producto a un carrito de compras

El siguiente ejemplo muestra cómo agregar un producto (o cualquier cosa) a una tabla de base de datos de forma asíncrona, utilizando AJAX y 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;
}
```

Para completar este ejemplo, actualice el código del lado del servidor para insertar estos datos en la base de datos. El código anterior asume que estás usando C # y tienes un archivo `Cart.cshtml` . Sin embargo, simplemente reemplace `cshtml` con `php` y escriba su propia lógica del lado del servidor usando el idioma de su elección.

Lea Empezando con ajax en línea: https://riptutorial.com/es/ajax/topic/1082/empezando-con-ajax

Capítulo 2: Devoluciones de llamada

Examples

Interpretación de errores con devolución de llamada "error"

Los errores, cuando sean administrados adecuadamente por el servidor, serán devueltos a su cliente con un código de estado HTTP específico diferente de 2xx (consulte la sección 10 de RFC 2616).

Se recomienda capturar globalmente los errores de su `$.ajaxSetup()` como se muestra en el siguiente ejemplo. Por lo tanto, todos los errores que provengan de sus llamadas ajax serán interpretados automáticamente desde la configuración de 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);
    }
});
```

También es posible que desee "sobrecargar" la devolución de llamada de `error` en un `$.ajax()` específico cuando está esperando un mensaje de error específico.

```
$.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
    }
});
```

Lea Devoluciones de llamada en línea: https://riptutorial.com/es/ajax/topic/7175/devoluciones-de-llamada

Creditos

S. No	Capítulos	Contributors
1	Empezando con ajax	acupajoe, adielhercules, Alessio Cantarella, Community, delete me, JhWebDevGuy, jignesh prajapati, MAZux, Menuka Ishan, Nicholas Qiao, TimTheEnchanter, Tolga Evcimen
2	Devoluciones de llamada	maxime_039