AJAX en JQuery

jQuery posee varios métodos para trabajar con Ajax. Sin embargo, todos están basados en el método $.ajax, por lo tanto, su comprensión es obligatoria.

El método $.ajax es configurado a través de un objeto, el cual contiene todas las instrucciones que necesita jQuery para completar la petición. Dicho método es particularmente útil debido a que ofrece la posibilidad de especificar acciones en caso que la petición haya fallado o no. Además, al estar configurado a través de un objeto, es posible definir sus propiedades de forma separada, haciendo que sea más fácil la reutilización del código.

```
$.ajax({
// la URL para la petición
url : 'prueba.php',

// la información a enviar
// (también es posible utilizar una cadena de datos)
data : { id : 123 },

// especifica si será una petición POST o GET
type : 'GET',

// el tipo de información que se espera de respuesta
dataType : 'text',

// código a ejecutar si la petición es satisfactoria;
// la respuesta es pasada como argumento a la función
success : function(data) {
    alert(data);
},

// código a ejecutar si la petición falla;
// son pasados como argumentos a la función
// el objeto de la petición en crudo y código de estatus de la petición
error : function(xhr, status) {
    alert('Disculpe, existió un problema.');
},

// código a ejecutar sin importar si la petición falló o no
complete : function(xhr, status) {
    alert('Petición realizada');
}
});
```

```
<?php
$param=$_REQUEST['id'];
echo $param;
?>
```

Si el servidor devuelve información que es diferente al formato especificado, el código fallará, y la razón de porque lo hace no siempre quedará clara debido a que la respuesta HTTP no mostrará ningún tipo de error.

El método $.ajax posee muchas opciones de configuración, y es justamente esta característica la que hace que sea un método muy útil, a continuación se muestran las más comunes:

- **async**. Establece si la petición será asíncrona o no. De forma predeterminada el valor es true. Debe tener en cuenta que si la opción se establece en false, la petición bloqueará la ejecución de otros códigos hasta que dicha petición haya finalizado.
- **cache**. Establece si la petición será guardada en la cache del navegador. De forma predeterminada es true para todos los dataType excepto para script y jsonp. Cuando posee el valor false, se agrega una cadena de caracteres anti-cache al final de la URL de la petición.
- **complete** Establece una función de devolución de llamada que se ejecuta cuando la petición está completa, aunque haya fallado o no. La función recibe como argumentos el objeto de la petición en crudo y el código de estatus de la misma petición.
- **context.** Establece el alcance en que la/las funciones de devolución de llamada se ejecutaran (por ejemplo, define el significado de this dentro de las funciones). De manera predeterminada this hace referencia al objeto originalmente pasado al método $.ajax.
- **data.** Establece la información que se enviará al servidor. Esta puede ser tanto un objeto como una cadena de datos (por ejemplo foo=bar&baz=bim)
- **dataType.** Establece el tipo de información que se espera recibir como respuesta del servidor. Si no se especifica ningún valor, de forma predeterminada, jQuery revisa el tipo de MIME que posee la respuesta.
- **error.** Establece una función de devolución de llamada a ejecutar si resulta algún error en la petición. Dicha función recibe como argumentos el objeto de la petición en crudo y el código de estatus de la misma petición.
- **jsonp.** Establece el nombre de la función de devolución de llamada a enviar cuando se realiza una petición JSONP. De forma predeterminada el nombre es *callback*
- **success.** Establece una función a ejecutar si la petición a sido satisfactoria. Dicha función recibe como argumentos la información de la petición (convertida a objeto JavaScript en el caso que dataType sea JSON), el estatus de la misma y el objeto de la petición en crudo.
- **timeout.** Establece un tiempo en milisegundos para considerar a una petición como fallada. traditional Si su valor es true, se utiliza el estilo de serialización de datos utilizado antes de jQuery 1.4. Para más detalles puede visitar api.jquery.com/jQuery.param.
- **type.** De forma predeterminada su valor es GET. Otros tipos de peticiones también pueden ser utilizadas (como PUT y DELETE), sin embargo pueden no estar soportados por todos los navegadores.
- **url**. Establece la URL en donde se realiza la petición. La opción url es obligatoria para el método $.ajax;

En caso que no quiera utilizar el método $.ajax, y no necesite los controladores de errores, existen otros métodos más convenientes para realizar peticiones Ajax (aunque, como se indicó antes, estos están basados el método $.ajax con valores pre-establecidos de configuración).

Los métodos que provee la biblioteca son:

- $.get() Realiza una petición GET a una URL provista.
- $.post() Realiza una petición POST a una URL provista.
- $.getScript() Añade un script a la página.
- $.getJSON() Realiza una petición GET a una URL provista y espera que un dato JSON sea devuelto.

```
jQuery.get( url [, data ] [, success ] [, dataType ] )
```

```
jQuery.post( url [, data ] [, success ] [, dataType ] )
```

```
$(document).ready(function(){
    $('#boton').on('click', enviar);
    function enviar() {

        $.get('prueba.php', {dato:1, dato2:2}, function(data){
            alert(data);
        });

        $.ajax({
            data: {dato:1, dato2:5},
            url: "prueba.php",
            type: 'GET',
            success: function() {
                alert( "Funcionó");
            },
            error: function() {
                alert( "Ha ocurrido un error" );
            }
        });

        $.post('prueba.php', {dato:1, dato2:2}, function(data){
            console.log(data);
        });
    }
});
```

```
<?php
$suma=$_GET['dato']+$_GET['dato2'];
echo $suma;
?>
```

El método GET debe ser utilizado para operaciones no-destructivas — es decir, operaciones en donde se esta “obteniendo” datos del servidor, pero no modificando. Por ejemplo, una consulta a un servicio de búsqueda podría ser una petición GET.

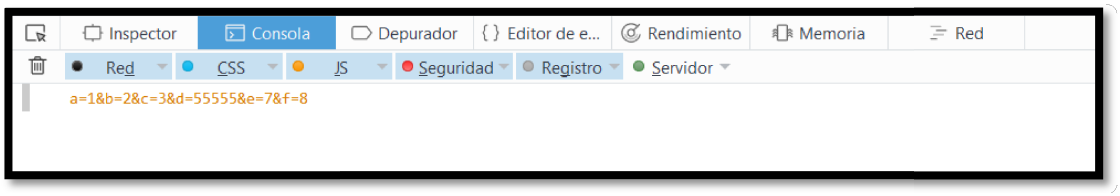
El método POST debe ser utilizado para operaciones destructivas — es decir, operaciones en donde se está incorporando información al servidor. Por ejemplo, cuando un usuario guarda un artículo en un blog, esta acción debería utilizar POST.

El método $.fn.load es el único que se puede llamar desde una selección. Dicho método obtiene el código HTML de una URL y rellena a los elementos seleccionados con la información obtenida. En conjunto con la URL, es posible especificar opcionalmente un selector, el cual obtendrá el código especificado en dicha selección.

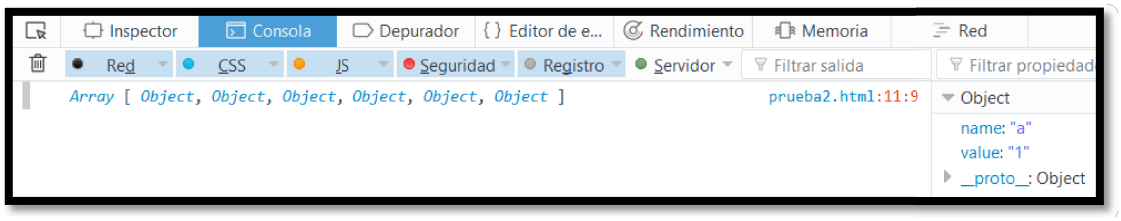
```
$(document).ready(function(){
    $('#boton').on('click', enviar);
    function enviar() {

        $('#newContent').load('/foo.html');

        $('#newContent').load('/foo.html #myDiv h1:first', function(html) {
            alert('Contenido actualizado');
        });
    }
});
```

A la hora de trabajar con formularios jQuery ofrece unos métodos que sirven de ayuda a la hora de enviar los datos, $.fn.serialize y $.fn.serializeArray.

```
$(document).ready(function(){
    $( "form" ).submit(function( event ) {
        console.log( $( this ).serialize() );
        event.preventDefault();
    });
});
```




```
$(document).ready(function(){
    $( "form" ).submit(function( event ) {
        console.log( $( this ).serializeArray() );
        event.preventDefault();
    });
});
```



Por último, jQuery ofrece una serie de funciones que responden a eventos de AJAX. Estas funciones no deben ser invocadas manualmente, se invocan automáticamente cuando se dispara su evento correspondiente.

- $.ajaxComplete()
- $.ajaxError()
- $.ajaxSend()
- $.ajaxStart()
- $.ajaxStop()
- $.ajaxSuccess()

A menudo, querrá ejecutar una función cuando una petición haya comenzado o terminado, como por ejemplo, mostrar o ocultar un indicador. En lugar de definir estas funciones dentro de cada petición, jQuery provee la posibilidad de vincular eventos Ajax a elementos seleccionados.

```
$(document).ready(function(){
    $('#loading_indicator')
        .ajaxStart(function() { $(this).show(); })
        .ajaxStop(function() { $(this).hide(); });
});
```