Ajax con jQuery (I)

Este artículo no pretende enseñarte lo que es Ajax ni, por supuesto, lo que es jQuery. Si estás leyendo esto, doy por sentado que conoces, aunque sea por encima, estas tecnologías. Este artículo pretende ser un recopilatorio - recordatorio de sintaxis, para poder usar Ajax con jQuery. El popular framework de JavaScript nos ofrece varias maneras básicas de trabajar con Ajax, y aquí vamos a conocerlas todas. Verás cómo es muy fácil y rápido combinar estas dos tecnologías para obtener un resultado inmejorable. En este artículo y los siguientes aprenderemos todo lo que necesitamos saber sobre el uso de la tecnología ajax en jQuery.

CARGA DIRECTA DE CONTENIDOS

Esta es, sin duda, la forma más simple de trabajar con Ajax y jQuery. Se basa en el método load(). Lo que hace es llamar a un script externo y cargar directamente los contenidos que dicho script genere.

Este método, cuando llama a un script externno, puede, por supuesto, pasarle parámetros que dicho script puede necesitar para su operativa. Estos pueden enviarse por GET o por POST, según nos convenga. En realidad, vamos a ver que este es un método muy simple de usar, pero con toda la flexibilidad y potencia que necesitemos.

PASO DE PARÁMETROS POR GET

Para empezar a comprender su funcionamiento, vamos a ver el fichero index_load_get.php, cuyo listado es el siguiente:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Pruebas Ajax</title>
</head>
<body>
 Nombre:
 <input type="text" value="" id="campo_nombre">
 <br>
 Email:
 <input type="email" value="" id="campo_email">
 <br>
 <button id="envio">Enviar</button>
 <div id="capaDeDatos"></div>
 <script language="javascript" src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
 <script language="javascript">
  $(function(){
   $('#envio').on('click', function(){
    var cadena = "campo_nombre=" + $('#campo_nombre').prop('value');
    cadena += "&campo_email=" + $('#campo_email').prop('value');
    $('#capaDeDatos').load("leer_datos_get_load.php", cadena);
   });
  });
```

```
 </script>
</body>
</html>
```

La mayoría de este código es HTML puro y duro. Aunque le hemos puesto la extensión .php por convencionalismo, ni siquiera tiene una sóla línea de PHP. Observa que hay dos campos, donde el usuario puede teclear su nombre y correo electrónico. Como aquí no vamos a usar funciones de validación, u otras complejas, ni siquiera hemos necesitado encapsular esos campos en un formulario, así que tampoco hay etiquetas <form></form> (no vamos a necesitarlas). De esta forma simplificamos al máximo. Observa también que los campos no tienen atributo name. Esto es porque no vamos a enviarlos de la forma convencional. En cambio, si cuentan con el atributo id, para poder referenciarlos desde jQuery.

Ahora vamos a ver el código jQuery, en las líneas que aparecen resaltadas en el listado. Cuando se pulsa el botón de envío, se construye una variable con dos pares nombre-valor. El nombre es el que elijamos para enviar cada campo y el valor es el que tenga en ese momento el campo. Esto se hace en las dos líneas siguientes:

```
var cadena = "campo_nombre=" + $('#campo_nombre').prop('value');
 cadena += "&campo_email=" + $('#campo_email').prop('value');
```

La cadena queda construida como se haría si fueramos a enviar estos parámetros como parte de una query string, es decir, supongamos que has tecleado como nombre Paco y como correo paco@paco.com. La cadena queda así:

```
campo_nombre=Paco&campo_email=paco@paco.com
```

A continuación aplicamos el método load() a una capa que hay en el documento, a través de su identificador. Lo que hace este método es llamar al script que figura en el primer parámetro y le añade, en la URL de la llamada la cadena que hemos construido, y que figura en el segundo parámetro. El resultado devuelto por ese script lo coloca (lo carga, cómo indica el nombre del método) en la capa que hemos referenciado en el selector:

```
$('#capaDeDatos').load("leer_datos_get_load.php", cadena);
```

El script al que se llama (leer_datos.php) obedece al siguiente listado:

```
<?php
 $nombre_recibido = $_GET["campo_nombre"];
 $email_recibido = $_GET["campo_email"];
 $resultado = "El nombre es ".$nombre_recibido." y el correo es ".$email_recibido.".";
 echo $resultado;
?>
```

Como ves, es muy simple. Recupera los valores a través de variables que han llegado por $_GET[. Con esos valores construye una cadena literal y la incluye en una instrucción echo. Esta instrucción, en condiciones "normales" volcaría la cadena en pantalla pero, cómo el script ha sido llamado por ajax, lo que hace es volcar esa cadena al proceso llamante, es decir, al método load() de jQuery que, al recibir la cadena, la carga en el contenedor referenciado por el selector.

Prueba la operativa para ver cómo funciona. Pon en tu navergador una llamada por localhost al primer script. Por ejemplo, yo tengo ambos dentro de una carpeta llamada ajax, dentro de localhost, así que tecleo http://localhost/ajax/index_load_get.php.

Teclea un nombre y un email en los campos correspondientes y observa cómo, al pulsar el botón de envío, se coloca el contenido esperado en el contendor reservado al efecto. Podríamos haber añadido mecanismos de control, cómo comprobar previamente que los datos tuvieran contenido, o que el email se ajustara a formato, pero no lo hemos hecho porque el objetivo real de este ejercicio no es ese, sino comprobar cómo funciona el método load(). En una web real si habríamos colocado esos mecanismos de control, pero este no es el caso.

PASO DE PARÁMETROS POR POST

Hemos visto que para pasar parámetros por GET costruíamos una cadena con dichos parámetros, del estilo de:

```
campo_nombre=Paco&campo_email=paco@paco.com
```

Para pasar valores por POST, el segundo parámetro de load() tiene que ser un objeto del estilo de JavaScript, como el ejemplo que aparece a continuación:

```
{'campo_nombre':'paco', 'campo_email':'paco@paco.com'}
```

La forma de construirlo la vemos en index_load_post.php:

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
 <meta charset="UTF-8">
 <title>Pruebas Ajax</title>
</head>
<body>
 Nombre:
 <input type="text" value="" id="campo_nombre">
 <br>
 Email:
 <input type="email" value="" id="campo_email">
 <br>
 <button id="envio">Enviar</button>
 <div id="capaDeDatos"></div>
 <script language="javascript" src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
 <script language="javascript">
  $(function(){
   $('#envio').on('click', function(){
    $('#capaDeDatos').load("leer_datos_get_load.php", {'campo_nombre':$('#campo_nombre').prop('value'),
'campo_email':$('#campo_email').prop('value')});
   });
  });
 </script>
</body>
</html>
```

Presta especial atención a la línea resaltada. El script que recibe estos datos por POST y los procesa (con el mismo resultado que antes) es leer_datos_get_load.php:

```
<?php
 $nombre_recibido = $_POST["campo_nombre"];
 $email_recibido = $_POST["campo_email"];
 $resultado = "El nombre es ".$nombre_recibido." y el correo es ".$email_recibido.".";
 echo $resultado;
?>
```

CARGA ASÍNCRONA Y CARGA SÍNCRONA]

Todos los procesos basados en ajax funcionan, nativamente, en modo asíncrono. Esto quiere decir que, cuando se llama a un script externo (pasándole o no parámetros, según las necesidades), el script principal se sigue ejecutando. Esto tiene la ventaja de que, cómo usuarios, no nos encontramos con la página bloqueada si el script llamado tarda un cierto tiempo en hacer lo que tiene que hacer. Como contrapartida, el inconveniente es que, si el resto del código del script principal depende de los datos que deba devolver el script llamado, podrían no estar disponibles cuando sean necesarios. Vamos a ver un ejemplo. Vamos a modificar la página que hemos usado para que tenga una capa más, con el estado de la petición ajax. Cuando se pulse el botón Enviar se mostrará el texto Cargando... y cuando el proceso finalice se mostrara el texto DATOS CARGADOS. En este ejemplo vamos a usar envío por POST, pero lo que aquí vamos a ver nos vale también para envíos por GET. Observa el script index_asincrono.php:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Pruebas Ajax</title>
</head>
<body>
 Nombre:
 <input type="text" value="" id="campo_nombre">
 <br>
```

```
 Email:
 <input type="email" value="" id="campo_email">
 <br>
 <button id="envio">Enviar</button>
 <div id="capaDeEstado"></div>
 <div id="capaDeDatos"></div>
 <script language="javascript" src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
 <script language="javascript">
  $(function(){
   $('#envio').on('click', function(){
    $('#capaDeEstado').html("Cargando datos...");
    $('#capaDeDatos').load("leer_datos.php", {'campo_nombre':$('#campo_nombre').prop('value'),
'campo_email':$('#campo_email').prop('value')});
    $('#capaDeEstado').html("DATOS CARGADOS");
   });
  });
 </script>
</body>
</html>
```

Observa las líneas resaltadas. Se pretende que al pulsar el botón se muestre un mensaje que invita a esperar lo necesario, se procesa el ajax y, al acabar se pretende que se muestre un mensaje que avise de que se ha terminado. El script leer_datos.php incluye una demora forzada para que podamos apreciar la operativa. Mira el listado:

```
<?php
 $nombre_recibido = $_POST["campo_nombre"];
 $email_recibido = $_POST["campo_email"];
 $resultado = "El nombre es ".$nombre_recibido." y el correo es ".$email_recibido.".";
 sleep(3);
 echo $resultado;
?>
```

En la línea resaltada vemos la demora de tres segundos.

Si pruebas esto, verás que, al pulsar el botón, te aparece, casi inmediatamente, el mensaje que dice que los datos están cargados, aunque, en realidad, tardan tres segundos en aparecer. El primer mensaje aparece y desaparece tan rápido que no puedes ni verlo. Esta es la operativa asíncrona, cómo hemos dicho. El script principal se sigue ejecutando, aunque el script llamado por ajax no se haya completado. En este caso, eso es un problema, porque nos aparece el mensaje de que los datos están cargados cuando, realmente, no lo están aún.

Este es un ejemplo muy claro en el que necesitaríamos una operativa síncrona. Esto significa que, hasta que los datos no estén cargados (tarden lo que tarden) el resto del script (el cambio de mensaje, en este caso) no se ejecute hasta que el proceso ajax haya concluido. Para poder hacer esto necesitamos añadirle, al método load(), un tercer parámetro: una función de callback que contenga el código a ejecutar cuando el proceso ajax haya concluido. Mira el script index_sincrono.php:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Pruebas Ajax</title>
</head>
<body>
 Nombre:
 <input type="text" value="" id="campo_nombre">
 <br>
 Email:
```

```
 <input type="email" value="" id="campo_email">
 <br>
 <button id="envio">Enviar</button>
 <div id="capaDeEstado"></div>
 <div id="capaDeDatos"></div>
 <script language="javascript" src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
 <script language="javascript">
  $(function(){
   $('#envio').on('click', function(){
    $('#capaDeEstado').html("Cargando datos...");
    $('#capaDeDatos').load("leer_datos.php", {'campo_nombre':$('#campo_nombre').prop('value'),
'campo_email':$('#campo_email').prop('value')}, function(){
     $('#capaDeEstado').html("DATOS CARGADOS");
    });
   });
  });
 </script>
</body>
</html>
```

Ejecútalo en tu navegador. Verás que ahora sí funciona cómo se espera que lo haga. Observa en las líneas resaltadas cómo hemos montado la función callback y, en el cuerpo de la misma, hemos incluido el código que cambia el mensaje de estado.

ATENCIÓN. Una función callback en un proceso jQuery es aquella que se define cómo parámetro de un método (sea el que sea) y se empieza a ejecutar cuando dicho método ha completado la tarea que se espera de él, cómo hemos visto en el último ejemplo. Las funciones callback son un poderoso recurso de jQuery ampliamente utilizado. Si estás leyendo este artículo, entiendo que estás más o menos familiarizado con este concepto. No obstante, si lo deseas, puedes aprender sobre funciones callback en jQuery en este enlace. Aquí no vamos a entrar en ello, porque no está dentro del objetivo de este artículo.

Del mismo modo que hemos usado la operativa síncrona con el método POST, podemos usarla, cómo hemos dicho, mediante GET. Observa el listado index_sincrono_get.php:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Pruebas Ajax</title>
</head>
<body>
 Nombre:
 <input type="text" value="" id="campo_nombre">
 <br>
 Email:
 <input type="email" value="" id="campo_email">
 <br>
 <button id="envio">Enviar</button>
 <div id="capaDeEstado"></div>
 <div id="capaDeDatos"></div>
```

```
 <script language="javascript" src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
 <script language="javascript">
  $(function(){
   $('#envio').on('click', function(){
    $('#capaDeEstado').html('Cargando datos...');
    var cadena = "campo_nombre=" + $('#campo_nombre').prop('value');
    cadena += "&campo_email=" + $('#campo_email').prop('value');
    $('#capaDeDatos').load("leer_datos_sincrono_get.php", cadena, function(){
     $('#capaDeEstado').html('DATOS CARGADOS');
    });
   });
  });
 </script>
</body>
</html>
```

Observa las líneas resaltadas. Una vez más, hemos incluido la acción final en una función callback. El script que procesa los datos, forzando una demora para comprobar la operativa, lo tenemos listado en leer_datos_sincrono_get.php:

```
<?php
 $nombre_recibido = $_GET["campo_nombre"];
 $email_recibido = $_GET["campo_email"];
 $resultado = "El nombre es ".$nombre_recibido." y el correo es ".$email_recibido.".";
 sleep(3);
 echo $resultado;
?>
```

CONCLUYENDO]

En este artículo hemos conocido la forma más simple, aunque totalmente operativa y eficiente, de usar ajax con jQuery: el método load(). También hemos sentado conceptos cómo el envío por POST y por GET, y las operativas síncronas y asícronas: cómo se programan y cuando nos conviene usar cada una de ellas.

En el próximo articulo vermos más sistemas de comunicación ajax en jQuery.