**Capítulo 7. Ajax**

**7.1. Introducción**

El método XMLHttpRequest (XHR) permite a los navegadores comunicarse con el servidor sin la necesidad de recargar la página. Este método, también conocido como Ajax (*Asynchronous JavaScript and XML*), permite la creación de aplicaciones ricas en interactividad.

Las peticiones Ajax son ejecutadas por el código JavaScript, el cual envía una petición a una URL y cuando recibe una respuesta, una función de devolución puede ser ejecutada la cual recibe como argumento la respuesta del servidor y realiza algo con ella. Debido a que la respuesta es asíncrona, el resto del código de la aplicación continua ejecutándose, por lo cual, es imperativo que una función de devolución sea ejecutada para manejar la respuesta.

A través de varios métodos, jQuery provee soporte para Ajax, permitiendo abstraer las diferencias que pueden existir entre navegadores. Los métodos en cuestión son $.get(), $.getScript(), $.getJSON(), $.post() y $().load().

A pesar que la definición de Ajax posee la palabra XML, la mayoría de las aplicaciones no utilizan dicho formato para el transporte de datos, sino que en su lugar se utiliza HTML plano o información en formato JSON (*JavaScript Object Notation*).

En general, Ajax no trabaja a través de dominios diferentes. Sin embargo, existen excepciones, como los servicios que proveen información en formato JSONP (*JSON with Padding*), los cuales permiten una funcionalidad limitada a través de diferentes dominios.

# 7.2. Conceptos clave

La utilización correcta de los métodos Ajax requiere primero la comprensión de algunos conceptos clave.

### 7.2.1. GET vs. POST

Los dos métodos HTTP más comunes para enviar una petición a un servidor son GET y POST. Es importante entender la utilización de cada uno.

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. Por otro lado, las solicitudes GET pueden ser almacenadas en la cache del navegador, pudiendo conducir a un comportamiento impredecible si no se lo espera. Generalmente, la información enviada al servidor, es enviada en una cadena de datos (en inglés query string).

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. Por otro lado, este tipo de método no se guarda en la cache del navegador. Además, una cadena de consulta puede ser parte de la URL, pero la información tiende a ser enviada de forma separada.

### 7.2.2. Tipos de datos

Generalmente, jQuery necesita algunas instrucciones sobre el tipo de información que se espera recibir cuando se realiza una petición Ajax. En algunos casos, el tipo de dato es especificado por el nombre del método, pero en otros casos se lo debe detallar como parte de la configuración del método:

* text Para el transporte de cadenas de caracteres simples.
* html Para el transporte de bloques de código HTML que serán ubicados en la página.
* script Para añadir un nuevo script con código JavaScript a la página.
* json Para transportar información en formato JSON, el cual puede incluir cadenas de caracteres, arrays y objetos.

**Nota**A partir de la versión 1.4 de la biblioteca, si la información JSON no está correctamente formateada, la petición podría fallar. Visite [json.org](http://json.org/) para obtener detalles sobre un correcto formateo de datos en JSON.

Es recomendable utilizar los mecanismos que posea el lenguaje del lado de servidor para la generación de información en JSON.

* jsonp Para transportar información JSON de un dominio a otro.
* xml Para transportar información en formato XML.

A pesar de los diferentes tipos de datos de que se puede utilizar, es recomendable utilizar el formato JSON, ya que éste es muy flexible, permitiendo por ejemplo, enviar al mismo tiempo información plana y HTML.

### 7.2.3. Asincronismo

Debido a que, de forma predeterminada, las llamadas Ajax son asíncronas, la respuesta del servidor no esta disponible de forma inmediata. Por ejemplo, el siguiente código no debería funcionar:

var response;

$.get('foo.php', function(r) { response = r; });

console.log(response); *// indefinido (undefined)*

En su lugar, es necesario especificar una función de devolución de llamada; dicha función se ejecutará cuando la petición se haya realizado de forma correcta ya que es en ese momento cuando la respuesta del servidor esta lista.

$.get('foo.php', function(response) { console.log(response); });

### 7.2.4. Políticas de mismo origen y JSONP

En general, las peticiones Ajax están limitadas a utilizar el mismo protocolo (http o https), el mismo puerto y el mismo dominio de origen. Esta limitación no se aplica a los scripts cargados a través del método Ajax de jQuery.

La otra excepción es cuando se hace una petición que recibirá una respuesta en formato JSONP. En este caso, el proveedor de la respuesta debe responder la petición con un script que puede ser cargado utilizando la etiqueta <script>, evitando así la limitación de realizar peticiones desde el mismo dominio. Dicha respuesta contendrá la información solicitada, contenida en una función.

### 7.2.5. Ajax y Firebug

Firebug (o el inspector WebKit que viene incluido en Chrome o Safari) son herramientas imprescindibles para trabajar con peticiones Ajax, ya que es posible observarlas desde la pestaña Consola de Firebug (o yendo a Recursos > Panel XHR desde el inspector de Webkit) y revisar los detalles de dichas peticiones. Si algo esta fallando cuando trabaja con Ajax, este es el primer lugar en donde debe dirigirse para saber cual es el problema.

# 7.3. Métodos Ajax de jQuery

Como se indicó anteriormente, jQuery posee varios métodos para trabajar con Ajax. Sin embargo, todos están basados en el método $.ajax, por lo tanto, su comprensión es obligatoria. A continuación se abarcará dicho método y luego se indicará un breve resumen sobre los demás métodos.

Generalmente, es preferible utilizar el método $.ajax en lugar de los otros, ya que ofrece más características y su configuración es muy comprensible.

### 7.3.1. $.ajax

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. Puede visitar [api.jquery.com/jQuery.ajax](http://api.jquery.com/jQuery.ajax/) para consultar la documentación sobre las opciones disponibles en el método.

**Utilizar el método $.ajax**

$.ajax({

*// la URL para la petición*

url : 'post.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 : 'json',

*// código a ejecutar si la petición es satisfactoria;*

*// la respuesta es pasada como argumento a la función*

success : function(json) {

$('<h1/>').text(json.title).appendTo('body');

$('<div class="content"/>')

.html(json.html).appendTo('body');

},

*// 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');

}

});

**Nota**Una aclaración sobre el parámetro dataType: 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. Cuando esté trabajando con peticiones Ajax, debe estar seguro que el servidor esta enviando el tipo de información que esta solicitando y verifique que la cabecera Content-type es exacta al tipo de dato. Por ejemplo, para información en formato JSON, la cabecera Content-type debería ser application/json.

**Opciones del método $.ajax**

El método $.ajax posee muchas opciones de configuración, y es justamente esta característica la que hace que sea un método muy útil. Para una lista completa de las opciones disponibles, puede consultar [api.jquery.com/jQuery.ajax](http://api.jquery.com/jQuery.ajax/); 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 esta 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](http://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;

### 7.3.2.  Métodos convenientes

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.

Los métodos deben tener los siguientes argumentos, en orden:

* url La URL en donde se realizará la petición. Su valor es obligatorio.
* data La información que se enviará al servidor. Su valor es opcional y puede ser tanto un objeto como una cadena de datos (como foo=bar&baz=bim).Nota: esta opción no es valida para el método $.getScript.
* success callback Una función opcional que se ejecuta en caso que petición haya sido satisfactoria. Dicha función recibe como argumentos la información de la petición y el objeto en bruto de dicha petición.
* data type El tipo de dato que se espera recibir desde el servidor. Su valor es opcional. Nota: esta opción es solo aplicable para métodos en que no está especificado el tipo de dato en el nombre del mismo método.

**Utilizar métodos convenientes para peticiones Ajax**

*// obtiene texto plano o html*

$.get('/users.php', { userId : 1234 }, function(resp) {

console.log(resp);

});

*// añade un script a la página y luego ejecuta la función especificada*

$.getScript('/static/js/myScript.js', function() {

functionFromMyScript();

});

*// obtiene información en formato JSON desde el servidor*

$.getJSON('/details.php', function(resp) {

$.each(resp, function(k, v) {

console.log(k + ' : ' + v);

}); });

### 7.3.3. $.fn.load

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.

**Utilizar el método $.fn.load para rellenar un elemento**

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

**Utilizar el método $.fn.load para rellenar un elemento basado en un selector**

$('#newContent').load('/foo.html #myDiv h1:first', function(html) {

alert('Contenido actualizado');

});

# 7.4. Ajax y formularios

Las capacidades de jQuery con Ajax pueden ser especialmente útiles para el trabajo con formularios. Por ejemplo, la [extensión jQuery Form Plugin](http://jquery.malsup.com/form/) es una extensión para añadir capacidades Ajax a formularios. Existen dos métodos que debe conocer para cuando este realizando este tipo de trabajos: $.fn.serialize y $.fn.serializeArray.

**Transformar información de un formulario a una cadena de datos**

$('#myForm').serialize();

**Crear un array de objetos conteniendo información de un formulario**

$('#myForm').serializeArray();

*// crea una estructura como esta:*

[

{ name : 'field1', value : 123 },

{ name : 'field2', value : 'hello world' }

]

# 7.5. Trabajar con JSONP

En los últimos tiempos, la introducción de JSONP, ha permitido la creación de aplicaciones híbridas de contenidos. Muchos sitios importantes ofrecen JSONP como servicio de información, el cual se accede a través de una API (en inglés Application programming interface) predefinida. Un servicio particular que permite obtener información en formato JSONP es [Yahoo! Query Language](http://developer.yahoo.com/yql/console/), el cual se utiliza a continuación para obtener, por ejemplo, noticias sobre gatos:

**Utilizar YQL y JSONP**

$.ajax({

url : 'http://query.yahooapis.com/v1/public/yql',

*// se agrega como parámetro el nombre de la función de devolución,*

*// según se especifica en el servicio de YQL*

jsonp : 'callback',

*// se le indica a jQuery que se espera información en formato JSONP*

dataType : 'jsonp',

*// se le indica al servicio de YQL cual es la información*

*// que se desea y que se la quiere en formato JSON*

data : {

q : 'select title,abstract,url from search.news where query="cat"',

format : 'json'

},

*// se ejecuta una función al ser satisfactoria la petición*

success : function(response) {

console.log(response);

}

});

jQuery se encarga de solucionar todos los aspectos complejos de la petición JSONP. Lo único que debe hacer es especificar el nombre de la función de devolución (en este caso callback, según lo especifica YQL) y el resultado final será como una petición Ajax normal.

# 7.6. Eventos Ajax

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. Para una lista completa de eventos Ajax, puede consultar [docs.jquery.com/Ajax\_Events](http://docs.jquery.com/Ajax_Events).

**Mostrar/Ocultar un indicador utilizando Eventos Ajax**

$('#loading\_indicator')

.ajaxStart(function() { $(this).show(); })

.ajaxStop(function() { $(this).hide(); });