Mockjax (Abril 2014)

Raúl Andrés Ortiz Fuentes, 112508.

# INTRODUCCIÓN

E

l Mockjax es un framework desarrollado como un plugin de jQuery el cual permite emular a la perfección el comportamiento de un servidor.

# Objetos Mock

En la programación orientada a objetos, un mock es un objeto simulado o de sustitución encargado de imitar el comportamiento de un objeto real dentro de un contexto controlado. Una de las aplicaciones más comunes para este tipo de objetos es la de simular la respuesta de otro objeto como, por ejemplo, una petición AJAX. Su uso es muy controvertido dentro de las metodologías de desarrollo debido a la debilidad que presentan: si modificamos el objeto original, debemos recordar modificar el objeto simulado para no crear inconsistencias.

En Javascript, prácticamente el uso de estos objetos se limitan a emular la respuesta de un servidor frente a peticiones asíncronas, resultando útiles cuando no se dispone de una plataforma REST implementada.

Existen varios frameworks de este tipo actualmente a disposición de los desarrolladores. Algunos de los más reconocidos y completos son:

* JsMockito: http://jsmockito.org
* Mockjax: https://github.com/appendto/jquery-mockjax
* jqMock: http://code.google.com/p/jqmock
* JSMock: http://jsmock.sourceforge.net
* amok: http://code.google.com/p/amok
* Jack: http://boss.bekk.no/display/BOSS/Jack
* JJ: <http://github.com/nakajima/jj>

# Mockajax

*jquery.mockjax* es un plugin jQuery desarrollado por Jonathan Sharp cuya finalidad es ofrecer una herramienta para realizar pruebas sobre peticiones AJAX con el menor impacto posible sobre el código original. Para ello, emula el comportamiento del servidor interceptando las peticiones y devolviendo aquellos datos que hayamos previamente definido.

Ventajas

1. Perfectamente integrado con jQuery.
2. Intercepta todas las peticiones AJAX para emular una respuesta por parte del servidor.
3. Amplio conjunto de opciones configurables para el servidor emulado.
4. Sintaxis elegante al estilo jQuery.

Desventajas

1. Requiere de jQuery.
2. Monitoriza únicamente las llamadas AJAX.
3. *Vertical-align*

# Integrando Mockjax

Para comenzar a utilizar esta herramienta solo se tiene que descargar el archivo correspondiente y añadirlo al código en cuestión, como se muestra a continuación:

<**script** src="http://github.com/appendto/jquery-mockjax/raw/master/jquery.mockjax.js">  
</**script**>

Por lo general, una petición AJAX en jQuery tiene la siguiente estructura:

*$.ajax({*

*type : "GET",*

*url : "/foo/bar.html",*

*data : {*

*id : 1,*

*name : 'Carlos Benitez',*

*pass : '1234567890'*

*},*

*success :* ***function****( data ){*

*console.log( "First Method Data Saved: " , data );*

*}*

*});*

Aquí es donde comienza ser interesante el contar con un plugin como Mockjax que emule la respuesta de un servidor interceptando las peticiones realizadas. Su inicialización más simple, sería la siguiente:

*$.mockjax({*

*url : '/foo/bar.html',*

*responseText : 'Server Response Emulated'*

*});*

Una vez iniciado, el plugin localizará todos los métodos AJAX jQuery ($.ajax, $.get, $.post, etc…) para comprobar la URL a la que realizan la petición con aquellas definidas en el mock. En caso de coincidir, el plugin intercepta la petición y ejecuta su contenido antes de devolver el control al manejador nativo jQuery.ajax. Si no coinciden las URLs, la petición se devuelve a $.ajax para su ejecución normal.

1. *Definiendo las URLs*

Existen varias formas de asignar la URL que nuestro mock debe interceptar. Desde una simple cadena:

*$.mockjax({*

*url: '/url/to/rest-service'*

*});*

A expresiones regulares que definan rangos:

*$.mockjax({*

*// Matches /data/quote, /data/tweet but not /data/quotes*

*url: /^\/data\/(quote|tweet)$/i*

*});*

Podemos incluso asignar comodines para una mayor cobertura:

*$.mockjax({*

*// Matches /data/quote, /data/tweet etc.*

*url: '/data/\*'*

*});*

*$.mockjax({*

*// Matches everything!*

*url: '\*'*

*});*

1. *Definiendo la respuesta*

El segundo aspecto importante es definir la respuesta que esperamos del servidor. Ésta puede ser tanto una cadena (responseText) como una estructura XML (responseXML).

Un ejemplo de respuesta en texto plano podría ser:

$.mockjax({

url: '/restful/api',

responseText: 'A text response from the server'

});

Sin embargo, si se necesita devolver una gran cantidad de datos, esta solución de texto inline, no resulta muy manejable. Para esos casos, se cuenta con la posibilidad de cargar los datos directamente con un proxy:

*$.mockjax({*

*url: '/restful/api',*

*proxy: '/mocks/data.json'*

*});*

El otro escenario común es cuando precisamos de un callback donde tengamos acceso a los posibles argumentos utilizados en la petición. Volviendo al ejemplo del principio, tendríamos:

*$.mockjax({*

*url: '/foo/bar.html',*

*response : function( settings ){*

*var data = settings.data;*

*this.responseText = data;*

*}*

*});*

*$.get( '/foo/bar.html', {*

*id : '1',*

*name : 'Carlos Benitez',*

*pass : '1234567890'*

*}, function( data ){*

*console.log( "Data Saved: " + data );*

*});*

En esta ocasión, la respuesta de nuestro mock corresponderá a todos los argumentos enviados:

//Data Saved: Object { id=1, name="Carlos Benitez", pass="1234567890"}

Si se observa con atención el ejemplo anterior, se ha utilizado la variable settings para extraer los argumentos. Esta variable contiene además, todos los datos relativos al mock:

* *data.* Los argumentos.
* *type.* El tipo de petición (GET, POST).
* *url.* La URL monitorizada.
* *success.* Función para el evento success.

# Conclusión

Si trabajamos con jQuery y necesitamos controlar las respuestas de nuestras peticiones a servidor, este plugin es sin duda alguna la opción perfecta. La gran cantidad de opciones configurables permite simular todos los supuestos que pueden darse durante la ejecución asíncrona de nuestro código ahorrándonos la tediosa tarea de requerir una arquitectura REST implementada.

**Primer autor** Raúl Andrés Ortiz Fuentes, 112508, Aplicaciones Web Avanzadas.