JQUERY MOCKJAX

WILSON ENRIQUE SALAZAR SIERRA **COD: 092502**

Es un plugin de JQuery desarrollado por Jhonatan 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 intersectando las peticiones y devolviendo aquellos datos que previamente hemos definido. Es soportado por todas las versiones superiores a la 1.3.2 de JQuery, siendo probado en los principales exploradores como Explorer 6, Firefox 3.3, Safary 3, Chrome 3 y Opera 9.

Este plugin proporciona una forma no invasiva para simular peticiones AJAX en toda la aplicación, adicional a esto es posible simular completamente el ciclo de petición y respuesta sin ningún tráfico de red y sin realizar ningún tipo de producción.

El uso de JQuery Mockups incluye:

- incluye jquery.mockjax.js jQuery Plugin
- Opcionalmente incluye json2.js de forma nativa compatible con el navegador.
- Incluye jquery.xmldom.js Simulado Xml.

La mayoría de los desarrolladores back-end están familiarizados con la terminología de mocking objects, para pruebas a pesar de ser bastante nuevo. Gran parte del desarrollo que appendTo hace se centra en el desarrollo de aplicaciones para usuario atada a los servicios Web RESTful. Como tal estamos en condiciones a las especificaciones del contrato de servicios y formato de los datos en el inicio de un proyecto y el desarrollo de la interfaz de front-end con los datos simulados, mientras que el equipo de back-end construye los servicios de producción.

El plugin fue desarrollado originalmente por appendTo de nuevo en marzo de 2010 y el equipo ha estado usando en todos sus proyectos desde entonces.

Mockjax consiste en dos métodos, uno para establecer la simulación, y otro para eliminarla. A continuación relacionaremos unos ejemplos.

VENTAJAS

1. **Iteración rápida en los servidores:**

   Cuando se desarrolla en un servidor local, la respuesta ajax aproximadamente se demora alrededor de los 100-300 milisegundos, incluso puede demorarse un poco más. Cuando lo trabajamos en un servidor externo podríamos ver diferentes tiempos de respuesta que pueden estar entre los 500 milisegundos y más de un segundo. Si usted está tratando de

desarrollar rápidamente una aplicación que toma un segundo por respuesta, va a llevar un tiempo ejemplar, por medio de la simulación de Mockjax jQuery la respuesta será mucho más efectiva y eficaz.

2. Disponibilidad de desarrollo en diferentes lugares:

Hay una serie de razones por las que un punto final de la API podría no estar disponible, por ejemplo necesita trabajar sobre su aplicación en un determinado momento o lugar sin tener acceso a WIFI como un avión u otro lugar en particular. Esto puede ser muy frustrante, e incluso podría hacer que usted pierda una fecha límite, de avance o entrega. Mockjax jΘuery le permite seguir trabajando sin ningún tipo de restricción como la anterior mencionada.

3. Servidor no disponible o lento en respuesta:

¿Cómo funciona su aplicación? Esta maneja un servidor remoto que se encuentra lento o no está disponible? Con la simulación de Mockjax jQuery, esto sería bastante fácil de manipular, teniendo control del punto final. Adicional nos permite realizar la simulación de asignarle características especiales al servidor.

EJEMPLOS

- En este ejemplo, vamos a configurar una simulación básica. Al ejecutar el código de ejemplo, nos dará como resultado o salida 'su cerveza al azar es Sam Adams. Nos daremos cuenta de que se ha iniciado la sesión Mockjax: 'GET SIMULACRO: / api / cerveza / random'. Esto nos permite saber que estamos simulando.

```
$.mockjax({
  url: '/api/beer/random',
  responseTime: 150,
  responseText: {
   success        : true
   ,beerName       : 'Sam Adams'
 }
});

$.getJSON('/api/beer/random', function(resp){
 var html = (resp.success) ? 'Your random beer is ' + resp.beerName
: 'No beer today';
 $('#content').html(html);
});
```

- Mockjax también nos permite especificar una respuesta a la solicitud de llamada a cambio de un responseText específico. Esto nos permite definir la respuesta mediante programación. Este es un ejemplo funcional

```
$.mockjax(function(settings){
 // Try to match our API URL string
  var service = settings.url.match(/\/api\/beer\/get\/(.*)$/i);

 // If we find a match, handle the response
  if (service) {

    // Convert the string to a number
   var beer_id = Number(service[1]);

   // JavaScript counts from 0
   var response = ibjhb.mock.beer[beer_id - 1];

    // If the response doesn't exist, defer to making an ajax,
   // otherwise create our return JSON packet to be returned
   return (typeof response === 'undefined')
      ? undefined
     :   {
         responseText : {
             success  : true
            ,beer   : response
          }
       }
 }
 return;
});
```