AJAX © ADRINFOR S.L.

www.adrformacion.com © ADRINFOR S.L.
Héctor García González
www.adrformacion.com © ADRINFOR S.L.
Héctor García González

Indice

AJAX

Introducción

Hasta hace relativamente poco tiempo, las acciones que un usuario realizaba en una web (como enviar un formulario de búsqueda o pulsar un enlace) provocaban una petición hacia el servidor web, de modo que el navegador debía recargar la página con la respuesta a esta petición.

Aunque completamente funcional, esta técnica no era amigable al usuario, dado que los datos transferidos y la recarga de la página conllevaban una mayor lentitud al mostrar la nueva información. Además, en muchas ocasiones, la página resultante era muy similar a la que se encontraba cargada.

Este tipo de funcionamiento era frecuente usado en selectores relacionados:

- Selectores de Provincia - Municipio
- Filtros para búsquedas en catálogos por Categoría - Subcategoría del producto.

Este funcionamiento era muy molesto, por la lentitud del mismo. Como solución, aparecen diferentes técnicas con las que se mejoraba notablemente la interacción del usuario con la aplicación, suprimiendo la necesidad de recargar constantemente la página y realizando la transferencia de información de forma transparente al usuario. Con el uso de AJAX se consigue una mejora importante en los tiempos de respuesta de la página, evitando que el usuario se encuentre con una página vacía a la espera de respuesta a su petición.

Técnicamente, el nombre AJAX son el acrónimo de las palabras "Asynchronous Javascript And XML", o en castellano, Javascript y XML Asíncrono.

En un lenguaje algo más simplificado, podríamos decir que éste fue el nombre normalizado que se le asignó a un conjunto de técnicas mediante las que de forma asíncrona, se hacía posible la obtención de información de un servidor web sin necesidad de recargar la página. Estas técnicas se componen de varias tecnologías que interrelacionadas ofrecen utilidades ilimitadas:

- CSS y XHTML para crear la presentación
- El componente XMLHttpRequest para el intercambio asíncrono de la información con el servidor
- XML o JSon para el intercambio de la información entre el navegador web y el servidor

- JavaScript tanto para lanzar la petición como para manejar la información de vuelta por el servidor

Con jQuery el uso de estas tecnologías se simplifica considerablemente, facilitando la creación de aplicaciones que utilizan esta tecnología para realizar comunicaciones con servidores web, reduciendo la cantidad de código fuente necesaria para construirlas y simplificando el entendimiento del mismo.

El uso de AJAX mejora considerablemente la experiencia de los usuario con los elementos de una página ya que la transferencia de información es ajena al usuario dado que se realiza en un segundo plano, evitando así la necesidad de recargar la página.

Conceptos Básicos

Servidor web

Cuando realizamos una petición AJAX, ésta será enviada a un servidor web, que deberá devolvernos una respuesta a esta petición. Si dispones de un espacio web personal, podrás utilizarlo como destino de tus peticiones cargando tus páginas y scripts al mismo y utilizarlo como entorno de producción.

in embargo, si no dispones de ninguna cuenta de hosting puedes utilizar un entorno local para realizar tus pruebas. Existen diferentes soluciones con las que configurar un servidor web en un ordenador personal y disponer así de un entorno de pruebas en el que probar tu código.

Estas soluciones son diferentes dependiendo del sistema operativo sobre el que estemos trabajando. Los acrónimos de estas herramientas para las diferentes infraestructuras son:

Estos entornos están preparados para ser ejecutados sobre el sistema operativo Windows: Generalmente, suelen funcionar en las diferentes versiones del sistema operativo de Microsoft. Existen diferentes instalaciones WAMP, como por ejemplo Uniform Server o Bitnami (para Windows)

Se denomina MAMP al entorno con Apache, MySQL y PHP configurado sobre sistemas operativos Apple Macintosh, MAC OS X. Uno de los configuradores más conocidos y utilizados, es MAMP o Bitnami (para OS X)

Por último, se identifica con el acrónimo LAMP al entorno configurado para ejecutar un servidor web desde entornos Linux: Bitnami (para Linux).

Configurar un entorno WAMP

Vamos a instalar un servidor WAMP para poder disponer en nuestro propio equipo de un servidor web que responda a nuestras peticiones AJAX. De entre las diferentes distribuciones de entornos WAMP utilizaremos Uniform Server por su sencillez.

Además, Uniform Server es un programa "portable": no requiere del registro de Windows, y el resultado de su instalación puede ser copiado en una memoria USB para disponer de una copia de seguridad o incluso para copiar el resultado a otro ordenador con sistema operativo Windows.

Descargar

En la página principal del sitio aparecerá un enlace con el que poder descargar el programa. Haremos clic sobre éste para acceder a la página de descarga.

El enlace nos redirige al repositorio de Sourceforge, donde para descargar el programa, pulsaremos sobre el enlace "Download Coral_X_x_x.exe". Pulsaremos el enlace y esperaremos unos segundos a que comience la descarga.

El programa de instalación tiene un tamaño aproximado de 15 MB. En una conexión normal de ADSL, la descarga debería completarse en un minuto o menos.

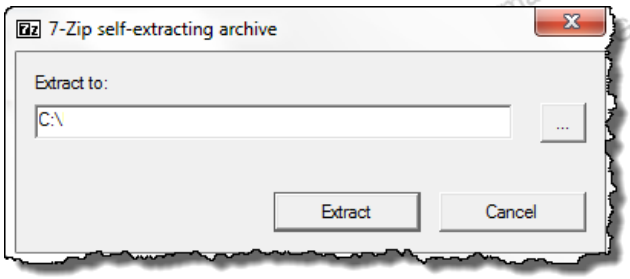
Para descargar Uniform Server, accederemos a su página oficial: http://www.uniformserver.com

Importante: Si la página de Uniform Server no funciona correctamente...

Si la página de Uniform Server no funciona correctamente puedes acceder directamente a la página de descarga desde el siguiente enlace
http://sourceforge.net/projects/miniserver/files/latest/download

Instalar

Una vez completada la descarga, ejecutaremos el programa. Éste, solicitará que se le indique una ruta donde realizar la extracción de Uniform Server. En este punto, podemos seleccionar cualquier ruta de nuestro equipo, por ejemplo, C:\

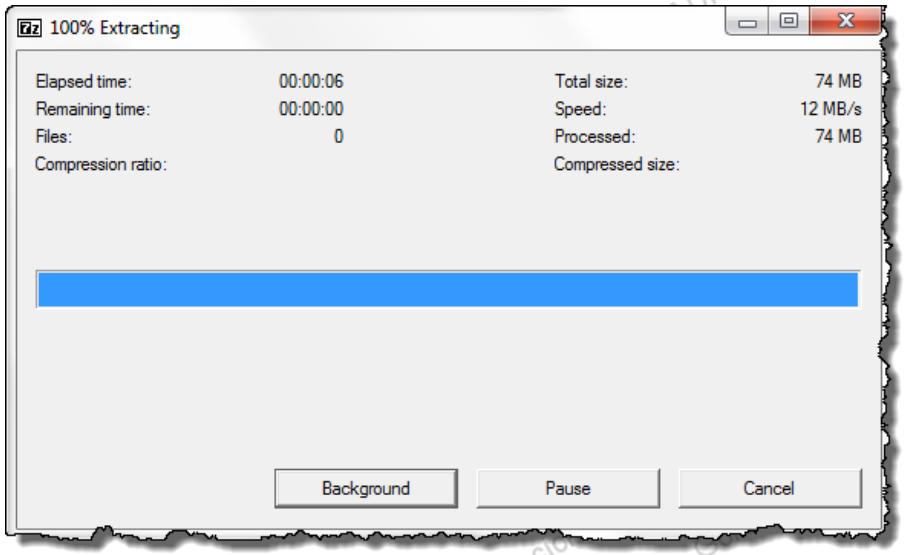


Con esta configuración, Uniform Server se extraerá dentro de una nueva carpeta "C:\UniServer" que contendrá todo lo necesario para configurar el entorno. Pulsaremos el botón "Extract" para confirmar la extracción.

Importante: Cuidado con la ruta indicada

Si en vez de C:\ indicas otra ruta, tu directorio web se referenciará a esa ruta.

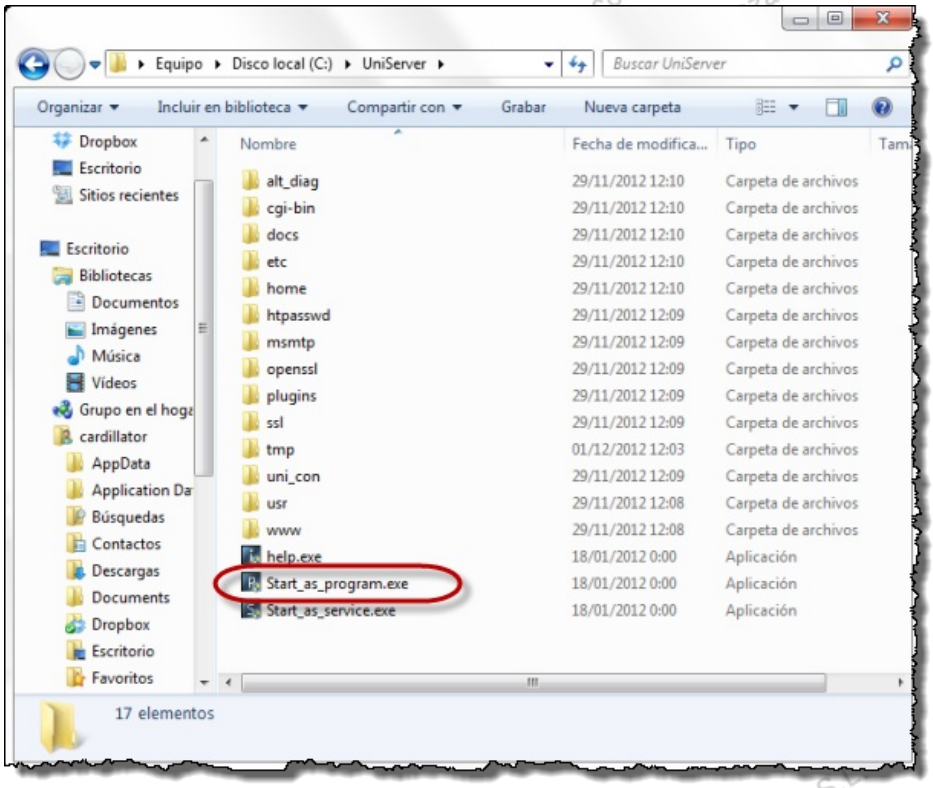
El tiempo de extracción dependerá de las prestaciones del ordenador en el que se ejecute. El programa de extracción indicará el tiempo estimado de extracción, así como una barra de progreso que indicará el porcentaje del proceso en el que se encuentre en cada momento.



Primera Ejecución

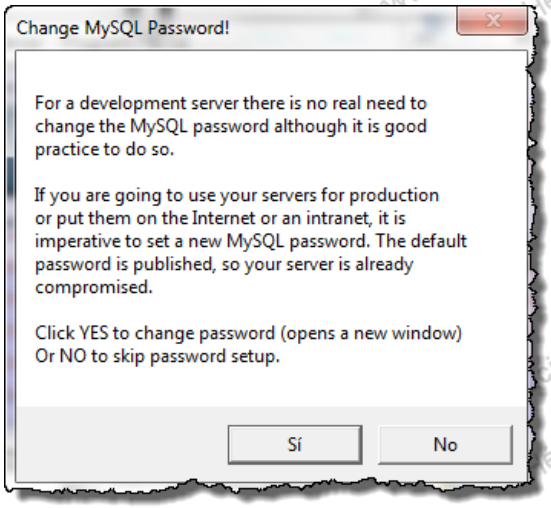
Una vez finalizada la extracción, accederemos a la carpeta indicada para la instalación. Si indicamos al ruta "C:\" encontraremos una nueva carpeta dentro de esta ruta, llamada "UniServer".

Dentro de la nueva carpeta creada, encontraremos el programa "Start_as_program.exe". Cada vez que vayamos a utilizar un entorno local - localhost - ejecutaremos este programa, para que se inicien todos los procesos necesarios para disponer de un servicio web en nuestro equipo.



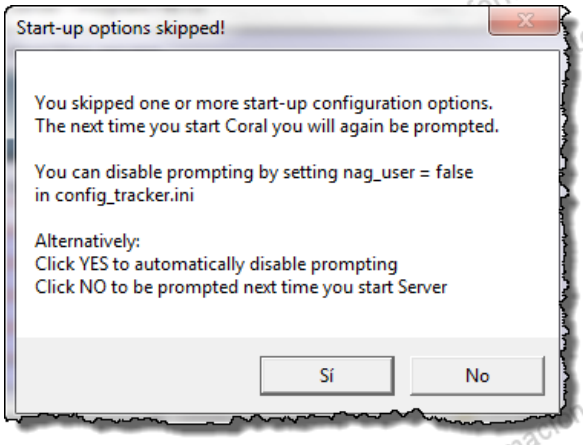
La primera vez que ejecutemos el programa, se nos mostrará un mensaje de aviso, ofreciendo la posiblidad de cambiar la contraseña por defecto para la base de datos.

En este ámbito no existe ninguna necesidad de modificar esta contraseña, dado que el entorno no va a estar disponible a visitantes externos, sino que solamente va a ser utilizado localmente desde el propio ordenador configurado como servidor. Por ello, pulsaremos el botón "No" para no realizar cambio de la contraseña.



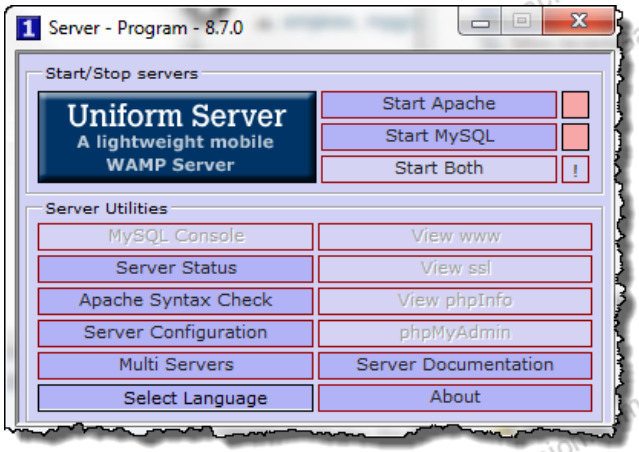
Si pese a la recomendación de no cambiar la contraseña decides hacerlo, recuerda apuntar la nueva contraseña ya que necesitarás este dato más adelante.

Si hemos omitido el proceso de cambio de contraseña - tal y cómo se recomienda - se mostrará otro aviso, recordándonos esta situación. Para evitar que cada vez que ejecutemos el programa se muestre este aviso pulsaremos el botón "Si" para deshabilitar estos avisos.

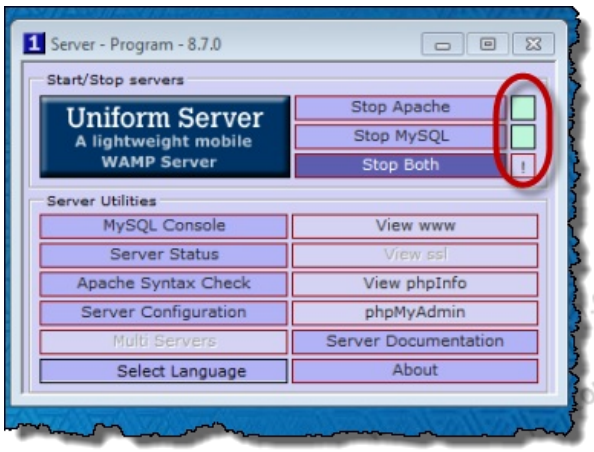


Iniciación de servicios

Una vez completadas estas acciones - y cada vez que ejecutemos el programa a partir de ahora - se nos mostrará la interface de gestión de Uniform Server. Por defecto, los servicios Apache y MySQL aparecerán en color rojo, indicando que no se encuentran activos.



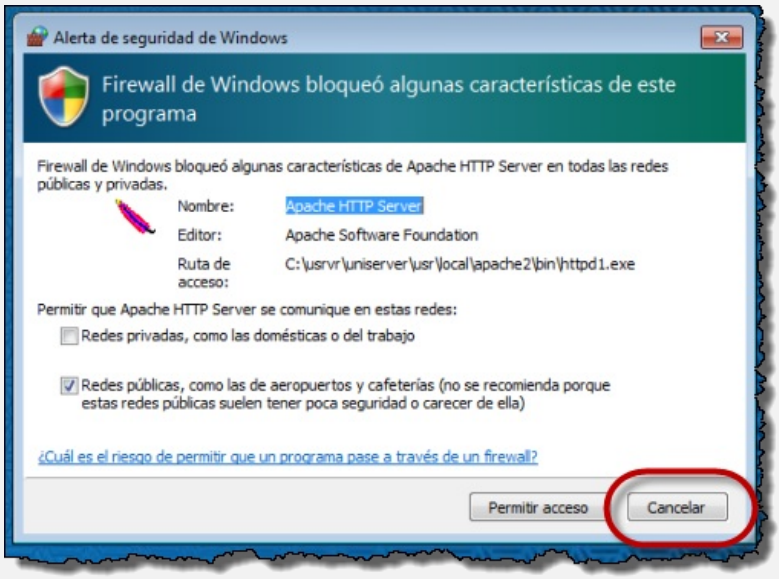
Para inicar los procesos, pulsaremos el botón "Start Both". Tras unos segundos, los servicios se pondrán en color verde, indicando que se encuentran activos.



Anotación: Si el Firewall te da guerra...

Si en tu equipo se encuentra activado el Firewall de Windows, aparecerá una ventana de notificación de bloqueo de Apache.

No es necesario concederle acceso a Internet al servidor Apache, con lo que podemos pulsar el botón "Cancelar" para no conceder este acceso.



Este proceso debe ser suficiente para que los servicios sean ejecutados correctamente. En algunas ocasiones, puede ocurrir que, al pulsar el botón de arranque de los servicios, alguno de ellos no pueda iniciar. Los servicios Apache y MySQL utilizan los puertos 80 y 3306 respectiamente para funcionar. Algunos programas existentes pueden utilizar alguno de estos puertos y en caso de estar ejecutandose en el momento en el que intentamos arrancar los servicios, los puertos estarán en uso y no permitirán arrancar los servicios. Si sospechas que algún programa hace uso de estos puertos, ciérralo antes de pulsar el botón de arranque de los servicios.

Un ejemplo de programa "conflictivo" es Skype. Skype utiliza el puerto 80 para sus comunicaciones. Si tienes Skype en ejecución al pulsar el botón de arranque de servicios, Apache no podrá inciar, y no mostrará su indicador en verde.

Para comprobar que el proceso ha funcionado completamente, accederemos a la dirección ***http://localhost/*** desde el navegador web.

Anotación: ¿Qué es LOCALHOST?

LOCALHOST es el nombre con el que nuestro ordenador se identifica a sí mismo, con lo que, cuando intentemos acceder a http://localhost/ el ordenador accederá a una página web alojada en nuestro ordenador y que solamente estará disponible desde el mismo.

¿Cómo funciona un servidor web?

Cuando accedemos a una dirección web o URL, el servicio web nos responderá mostrandonos una página web o, en caso de solicitar un recurso concreto, nos muestra ese elemento.

Por ejemplo, al acceder a la dirección http://www.adrformacion.com/ el servidor web que aloja esa página nos mostrará la página web "index.php" que contiene su carpeta raíz (carpeta principal).
Cuando nos referimos a un recurso concreto, por ejemplo http://www.adrformacion.com/img/cabeza.jpg el servidor web nos mostrará la imagen "cabeza.jpg" que se encuentra dentro de la subcarpeta "img/" del sitio.

Al ejecutar Uniform Server e iniciar los servicios, se configurará un servicio web, que al recibir alguna petición (cuando accedamos a la dirección http://localhost/) mostrará el recurso solicitado, siempre y cuando éste se encuentre dentro de la carpeta c:\UniServer\www\

Así, al acceder a http://localhost/index.php el servidor nos mostrará el archivo "index.php" de la carpeta c:\UniServer\www\ y actuará del mismo modo con todos los archivos que se encuentren dentro de c:\UniServer\www\

Cómo funciona un servidor web

Importante: ¿Cuándo será necesario iniciar Uniform Server?

Cada vez que queramos utilizar nuestro servidor local tendremos que ejecutar"Start_as_program.exe" y activar los servicios. Si no tenemos los servicios en ejecución no obtendremos respuesta al acceder a http://localhost/

Peticiones GET o peticiones POST

Cada vez que realizamos una petición al servidor, ésta puede ser de tipo GET o POST, dos métodos de envío diferentes soportados por el protocolo HTTP

Es recomendable utilizar el método GET para envío de peticiones cuyo objetivo sea la obtención de datos del servidor, como búsquedas o consulta de datos.

Con este tipo de peticiones, los parámetros enviados son incluidas en la URL, formando lo que se denomina "query string".

Las peticiones GET pueden verse afectadas por la caché del navegador. Ten en cuenta este detalle si observas algún comportamiento extraño.

Las peticiones POST son recomendables para enviar peticiones que van a tener una acción sobre el servidor, como inserción de datos, modificación o eliminación.

Estas peticiones no se ven afectadas por la caché del navegador, y la información enviada no será incluida en la URL sino que se utilizarán otros métodos para el envío de la misma.

Tipos de Datos

Los métodos de jQuery que gestionan las peticiones AJAX permiten manejar la información recibida de varios modos, lo que ofrece mayor versatilidad a la hora de programar así como la posibilidad de manejar diferentes tipos de respuestas.

Algunos métodos de la librería son específicos para determinados tipos de datos, mientras que en otros, podremos indicar el tipo de dato esperado como respuesta a la petición:

Tipo de dato text

Cuando se configura la librería, o se indica en el método de llamada que el dato recibido va a ser de tipo texto, la función que controle la obtención de la respuesta recibirá un parámetro cuyo valor se corresponderá con la cadena de texto que el servidor devuelva a la petición realizada.

Tipo de dato html

En el caso de que se indique como tipo de dato esperado la opción "html", la respuesta a la petición será tratada como un bloque de código HTML. Este tipo de dato es útli para gestionar respuestas que se compongan de fragmentos de HTML, cuya finalidad sea ser incrustada directamente en la página.

Tipo de dato script

Al indicar como tipo de dato la opción script, la respuesta obtenida será tratada como un archivo de código Javascript, y procesada del mismo modo que si incuyéramos en la página un elemento de tipo script.

Tipo de dato json

El tipo de dato JSON procesará la respuesta dentro de un objeto, cargando la información del mismo a partir de la interpretación de la respuesta según este formato.

Es importante que la estructura devuelta por el servidor cumpla con formato JSON válido. Si la respuesta no cumpliera con este formato, la petición podrá provocar errores.

Tipo de dato jsonp

Al igual que el tipo de dato json, este tipo de dato esperará como respuesta del servidor una estructura de datos válida. La principal diferencia en este tipo de datos es que permitirá hacer peticiones a diferentes dominios, previniendo así los filtros de seguridad de los navegadores en contra de peticiones de dominio cruzado (Cross Site Scripting o XSS).

Si deseas saber qué es el Cross Site Scripting (XSS) puedes encontrar más información en la siguiente dirección: http://es.wikipedia.org/wiki/Cross-site_scripting

Tipo de dato xml
El tipo de dato xml esperará como respuesta un fragmento de texto en formato XML. Al igual que ocurría con los tipos de datos json y jsonp, la respuesta debe tener una estructura correcta, ya que de lo contrario provocará un error en la ejecución del código.

Formato JSON

Entre las tecnologías que podemos encontrar al realizar peticiones AJAX encontramos JSON. JSON es un formato estándar de texto, utilizado para el intercambio de información que permite la transferencia de datos de forma ágil y sencilla. El formato JSON es independiente del lenguaje utilizado tanto para su generación como para su lectura. Este formato es comunmente utilizado en las transferencias de información vía AJAX, siendo de gran utilidad para representar, en formato de texto símple, un objeto con sus propiedades.

La estructura de JSON se compone de un formato símple, lo que ha conllevado una gran generalización de su uso como alternativa al uso de XML en la generación de respuestas para peticiones AJAX. Entre los motivos que han llevado al uso generalizado de JSON se encuentra la facilidad de analizar una respuesta en este formato por medio de funciones JavaScript.

JSON, acrónimo de JavaScript Object Notation, es un formato ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.

Fuente: Wikipedia

JSON no solamente es utilizado como respuesta para peticiones web. De hecho, existen importantes componentes basados en este estándar, como las bases de datos NoSQL MongoDB y CouchBase.

¿Por qué utilizar JSON?

JSON está reemplazando el formato XML como principal formato de intercambio de datos a través de la web porque el formato JSON perite su fácil interpretación y su estructura se asemeja a las utilizadas en los principales lenguajes de programación. Además, este formato es más efeciente que el formato XML, siendo más compacto gracias a la no necesidad de uso de etiquetas de comienzo y cierre.

Estructura

Como comentamos anteriormente, el formato JSON permite el intercambio de datos con una estructura ligera. Un documento JSON se basa en tres estructuras:

- Una colección de pares atributo : valor (key : value)
- Objetos
- Arrays de datos

Un documento JSON siempre se encuentra encerrado entre llaves:

{ JSON-Data }

Es importante conocer la estructura de datos JSON ya que, en muchas ocasiones, será el formato que recibiremos en respuesta a peticiones AJAX

Existen multitud de herramientas online para validar la estructura de una cadena de texto JSON. Una de las más utilizadas y sencillas de utilizar es http://jsonlint.com/

Pares atributo : valor

La estructura JSON que da soporte a la información de un objeto engloba las propiedades de los mismos entre los símbolos "{" y "}", conteniendo para cada atributo del objeto los pares "atributo":"valor" y separando cada uno de estos atributos de otro por el símbolo ",".

Veamos un sencillo ejemplo de un objeto JSON, en el que se muestra información de una cita de una agenda:

```
{
  "titulo":"Cena de negocios",
  "lugar":"Madrid",
  "numAsistentes": 5
}
```

El valor de un atributo puede ser una cadena, un valor numérico, un booleano, un array (cadena de elementos) e incluso un objeto obteniendo así una estructura de objetos anidados.

Objetos

En JSON, un objeto es una colección de pares atributo : valor, separados por una coma y a su vez, entrecerrado entre llaves. Si observamos la estructura, un objeto dentro de JSON tiene la misma estructura que el propio bloque completo. Así, es posible definir una estructura algo más compleja de objetos anidados.

Vamos a ampliar el ejemplo anterior, añadiendo la información horaria como un nuevo objeto, conteniendo de forma separada en los atributos "hora", "minuto" y "segundo":

```
{
  "titulo":"Cena de negocios",
  "lugar":"Madrid",
  "numAsistentes": 5,
  "time": {
    "hora": "21",
    "minuto": "30",
    "segundo": "00"
  }
}
```


Arrays de datos

Para representar un array de datos (también llamado cadena de elementos o arreglo), indicaremos los datos del mismo entre los símbolos "[" y "]", separando los diferentes valores del array por el símbolo ",".

Así mismo, cada uno de los valores de un array puede ser tanto un nuevo array como un objeto, pudiendo así representar con un fragmento de código JSON casi cualquier elemento que se nos pueda ocurrir.

Veamos un ejemplo de código JSON, que representa la información de un usuario, en el que el objeto representado está compuesto por datos de todos los tipos posibles.

```
{"usuario":
  {
    "infoPersonal": "John Doe",
    "picture": "http://lorempixel.com/180/220/people/Aprendiendo-jQuery/",
    "infoContacto": {
      "email": "sample@mail.com",
      "telefono": [
        {"tipo": "movil", "numero": "666555444"},
        {"tipo": "fijo", "numero": "555444666"}
       ],
      "redesSociales": [
        {"red": "twitter", "perfil": "@perfil"},
        {"red": "facebook", "perfil": "perfil"}
       ]
    },
    "valid": true
  }
}
```

Esta estructura ofrece un ejemplo avanzado de datos en JSON, mostrando un objeto (usuario) con cuatro atributos (infoPersonal, picture, infoContacto y valid), uno de ellos a su vez es otro objeto (infoContacto) y, entre sus valores, encontramos dos arrays de objetos (telefono, redesSociales).

El formato JSON

Política de Seguridad en peticiones AJAX

Pese a la gran utilidad del uso de esta tecnología nos encontramos con algunas limitaciones en el uso de la misma. Principalmente estas limitaciones vienen ocasionadas por motivos de seguridad.

La mayor limitación con la que nos encontraremos al utilizar AJAX esl con la política "Same Origin Policy" (Política del mismo origen) de los navegadores. Esta política de seguridad limita que las peticiones AJAX que realice una página únicamente sean posibles a recursos del mismo dominio, subdominio o protocolo utilizado en la carga de la misma.

En informática, la política del mismo origen es una medida importante de seguridad para scripts en la parte cliente (casi siempre javascript). Esta política viene del navegador web Netscape 2.0, y previene que un documento o script cargado en un "origen" pueda cargarse o modificar propiedades del documento desde un "origen" diferente. Se trata de uno de los conceptos de seguridad más importantes de los navegadores modernos.

Fuente: Wikipedia

Puedes obtener más información sobre la Política del mismo origen desde este enlace: http://es.wikipedia.org/wiki/Pol%C3%ADtica_del_mismo_origen

El método $.ajax() para realizar solicitudes asíncronas

Vamos a ver cómo funciona el método .ajax() para realizar slicitudes asíncronas con jQuery. Este método engloba todas las peticiones AJAX que puede realizar jQuery. Por ello, ofrece una amplia personalización que , siendo por un lado una importante característica, conlleva a su vez una mayor complejidad en su uso. Vamos a ver todas las opciones que ofrece este método, así como ejemplos de su funcionamiento para comprender el comportamiento del mismo según la información pasada al mismo.

La llamada más básica al método .ajax() será sin ningún argumento:

```
$.ajax();
```

En este caso, dado que no se ha indicado ningún parámetro al método, éste utilizará la configuración por defecto indicada a jQuery por medio del método $.ajaxSetup().

Sin embargo, la forma más habitual de realizar llamadas al método es proporcionándole a través de parámetros del método las opciones a utilizar. Esta llamada se realiza con un único parámetro en forma de objeto anónimo en el que cada una de las propiedades del objeto hará referencia a una opción, a partir del valor indicado como nombre de la propiedad.

```
$.ajax({
  // URL a la que realizar la petición
  url : 'target.php',

  // tipo de petición: GET o POST
  type : 'GET',

  // información a enviar en la petición
  data : { id : 123 },

  // tipo de respuesta esperada
  dataType : 'json',

  // función controladora de respuesta, que recibe como parámetro
  // la respuesta obtenida
  success : function(json) {
    console.log(json);
  }
});
```

Otra forma de realizar esta petición es almacenar en una variable el objeto anónimo, y posteriormente proporcionarle ésta al método:

```
var requestInfo = {
  // URL a la que realizar la petición
  url : 'target.php',

  // tipo de petición: GET o POST
  type : 'GET',

  // información a enviar en la petición
  data : { id : 123 },

  // tipo de respuesta esperada
  dataType : 'json',

  // función controladora de respuesta, que recibe como parámetro
  // la respuesta obtenida
  success : function(json) {
    console.log(json);
  }
};

$.ajax( requestInfo );
```

Opciones del método $.ajax

En el ejemplo anterior hemos visto la forma en la que indicar las opciones al método $.ajax(), haciendo la llamada al mismo con varias opciones habituales. A la hora de hacer una llamada al método no es necesario indicar todas las opciones, sino solamente aquellas que necesitemos utilizar.

Opciones Principales

Vamos a ver las principales opciones que podremos indicar en una llamada al método .ajax() para realizar una petición al servidor. Con estas opciones podremos especificar los detalles principales que definirán la petición: URL a la que realizar la petición, tip de petición, información a enviar, etc.

Pese a que todas las opciones de una petición son importantes este subconjunto son las que detallarán los aspectos básicos de la mismas. Casi todas nuestras peticiones contendrán, como mínimo, estas opciones.

Opción url

El parámetro url indicará al método la URL a la que tendrá que hacer la petición. El valor por defecto de este parámetro corresponderá con la url de la página actual

```
 $.ajax({
   // URL a la que realizar la petición
   url : 'target.php'
});
```

Opción type

En este parámetro podremos indicar el tipo de petición HTTP a realizar, si será una petición GET o petición POST. Para indicar uno u otro método, estableceremos este atributo con las cadeas "GET" o "POST". Si no se especifica un tipo, el valor por defecto utilizado para la petición será "GET".

Petición POST

```
 $.ajax({
   // URL a la que realizar la petición
   url : 'target.php',
   // tipo de petición
   type : 'POST'
});
```

Petición GET

```
 $.ajax({
   // URL a la que realizar la petición
   url : 'target.php',
   // tipo de petición
   type : 'GET'
});
```

Opción data

En el parámetro data indicaremos los datos que deseemos enviar al servidor cuando se realice la petición. Este parámetro podrá ser aceptado en dos formas:

- Un objeto anónimo, cuyos atributos se enviarán como nombres de los parámetros de la llamada, y los valores de los mismos como su valor
- Una cadena de texto, en formato cadena de consulta o "query string", en la que una única cadena de texto contiene una combinación de varios atributos con sus valores, separados estos por el símbolo "=" y entre ellos por el símbolo "&". Ejemplo: nombreparametro1=valor1&nombreparametro2=valor2

Una forma fácil de enviar los datos de un formulario en una petición AJAX es utilizando el método de jQuery .serialize() con el que obtener una cadena de consulta a partir de los elementos de formulario que se localicen tanto en la selección sobre la que se aplique como en sus descendientes. Este método será muy útil para realizar peticiones AJAX y utilizar el valor devuelto por el mismo como parámetro "data" en las llamadas al método .ajax().

Datos como objeto anónimo

```
 var userStatus="online";
var userName="Nombre Usuario";
$.ajax({
   // URL a la que realizar la petición
   url : 'target.php',
   // tipo de petición
   type : 'POST',
   // datos enviados en la petición
   data: {status: userStatus, name: userName}
});
```

Datos como cadena de consulta

```
 $.ajax({
   // URL a la que realizar la petición
   url : 'target.php',
   // tipo de petición
   type : 'POST',
   // datos enviados en la petición
   data: "status=online&name=Nombre%20Usuario"
});
```

Datos de un formulario serializados

```
 $.ajax({
   // URL a la que realizar la petición
   url : 'target.php',
   // tipo de petición
   type : 'POST',
   // datos enviados en la petición
   data: $('#form0').serialize()
});
```

Opción timeout

La opción timeout permite definir el tiempo tras el cual una petición no respondida será considerada como fallida.

Al indicar esta opción hemos de tener en cuenta que el valor debe definirse en milisegundos, que serán contados desde el momento en que se realice la llamada al método .ajax(). Si no se dispone de conexión a internet, es posible que la solicitud se considere como fallida antes de que transcurra el tiempo indicado en esta opción.

En el navegador Firefox 3.0 y versiones posteriores, las peticiones de tipo "script" y "jsonp" no pueden ser controladas con este valor.

Opción dataType

En esta opción indicará al método el tipo de dato que se espera obtener como respuesta desde el servidor.

Como valor, podrá aceptar cualquiera de los tipos de datos soportados: text, html, script, json, jsonp o text.

Si no se indica un valor para esta opción, el método intentará hacer un análisis inteligente de la respuesta y determinar el tipo de dato al que se corresponde.

Respuesta un script

```
$.ajax({
  // URL a la que realizar la petición
  url : 'target.php',
  // tratará la respuesta como si de un archivo js se tratara,
  //ejecutándolo al completar la petición
  dataType: 'script'
});
```

Respuesta objeto JSON

```
$.ajax({
  // URL a la que realizar la petición
  url : 'target.php',
  // tipo de respuesta esperada
  dataType: 'json'
});
```

Respuesta objeto JSON a partir de una solicitud jsonp

```
$.ajax({
  // URL a la que realizar la petición
  url : 'target.php',
  // tipo de respuesta esperada
  dataType: 'jsonp json'
});
```


Opciones de Control de la respuesta

Este otro grupo de opciones controlarán la respuesta a una petición. Como resultado a estas peticiones encontraremos dos escenarios diferentes:

- La petición ha sido realizada con éxito.
- La petición no se ha podido realizar o ha ocurrido algún error durante la misma.

Cada escenario deberá controlarse por separado, dado que las aplicaciones que creemos que utilicen esta tecnología deberán tener un comportamiento diferente si una petición no obtiene respuesta frente a una petición que sí que se puede procesar correctamente.

Opción success

Esta opción aceptará una función, que se ejecutará cuando la petición se complete con éxito.

La función recibira tres parámetros:

- El primero se corresponderá a los datos devueltos por el servidor en respuesta a la petición. Esta respuesta deberá cumplir con el formato indicado en la opción dataType.
- El segundo, una cadena de texto, correspondiente al estado. Entre sus posibles valores están:
  - abort
  - error
  - notmodified
  - parsererror
  - success
  - timeout
- El tercero, contendrá el objeto jqxHR (el objeto jQuery XMLHttpRequest) encargado de realizar la petición.

No será necesario que la función indicada espere los tres parámetros. Solamente en caso de que la misma vaya a utilizarlos será necesario indicarlo en la misma.

Petición que procesará una respuesta json

Cuando en la petición se ha establecido el tipo de dato (dataType) json, la función que conrolará el completado de la misma recibirá como primer parámetro un objeto a partir de la interpretación del código JSON devuelto por el servidor.

Vamos a suponer que, como respuesta a la petición, el servidor nos devuelve el siguiente código:

```
{
  "result":"1",
  "message":"Su petici&oacute;n se ha realizado correctamente",
  "reloadPage": "http://www.google.com/"
}
```

Veamos cómo definiríamos la llamada al método .ajax() para que, además de obtener el código, sea capaz de procesar esta respuesta. La función nos mostrará una alerta con el texto del mensaje devuelto o nos avisará de que ha ocurrido un error (si el valor de respuesta de "result" no es igual a "1") y nos redirigirá a la página indicada en el atributo "reloadPage" del objeto JSON de respuesta.

```
$.ajax({
  // URL a la que realizar la petición
  url : 'target.php',
  // tratará la respuesta como si de un archivo js se tratara,
  //ejecutándolo al completar la petición
  dataType: 'json',
  //función que controla la respuesta devuelta por el servidor
  success: function(jsonData, status, jqxHR){
    if(jsonData.result == "1"){
      alert(jsonData.message);
      document.location.href = jsonData.reloadPage;
    }
    else{
      alert("Su petición no ha podido ser procesada correctamente");
    }
  }
});
```

Petición que procesará una respuesta HTML

Las peticiones que esperan como respuesta un texto en formato HTML son una forma fácil de recoger el código HTML devuelto por el servidor para, posteriormente, incrustarlo dentro de algún contenedor en la página.

```
$.ajax({
  // URL a la que realizar la petición
  url : 'target.php',
  // tipo de respuesta esperada
  dataType: 'html',
  //función que controla la respuesta devuelta por el servidor
  success: function(htmlData, status, jqxHR){
    $("#infoContainer")
      .empty()
      .append(htmlData);
  }
});
```

Opción error

Al igual que la opción success, la opción error aceptará una función. En este caso, dicha función será ejecutada cuando se produzca un error en la petición.

Esta función también recibirá tres parámetros:

- El objeto jqxHR (el objeto jQuery XMLHttpRequest) utilizado para realizar la petición
- Una cadena de texto, que contendrá una descripción del error:
  - timeout
  - error
  - abort
  - parsererror
- Si el error es producido por un fallo en la petición HTTP, este parámetro contendrá la descripción del estado HTTP de respuesta.

A pesar de los parámetros de la función, la mejor opción para conocer el motivo por el que se ha producido un error es combinar el valor de las propiedades "status" y "responseText" del objeto jqxHR junto con el segundo parámetro enviado a la función controladora del error:

```
$.ajax({
  // URL a la que realizar la petición
  url : 'target.phpogg',
  // tipo de respuesta esperada
  dataType: 'json',
  //función que controla la respuesta devuelta por el servidor
  success: function(htmlData, status, jqxHR){
  },
  error: function(jqXHR, exception) {
    if (jqXHR.status === 0) {
      alert('No conectado.\n Verificar la conexión a Internet.');
    } else if (jqXHR.status == 404) {
      alert('Página no encontrada [404]');
    } else if (jqXHR.status == 500) {
      alert('Error interno del servidor [500].');
    } else if (exception === 'parsererror') {
      alert('Elrror al procesar la respuesta JSON.');
    } else if (exception === 'timeout') {
      alert('Tiempo máximo de respuesta superado.');
    } else if (exception === 'abort') {
      alert('Petición AJAX abortada.');
    } else {
      alert('Error no controlado.\n' + jqXHR.responseText);
    }
  }
});
```

Si se realiza una petición de tipo jsonp (para peticiones de dominio cruzado o Cross-domain) la función no será ejecutada.

Opción complete

Esta tercera opción permitirá indicar una función que se ejecutará cuando la petición finalice, independientemente de que ésta haya resultado exitosa o fallida.

En este caso, la función indicada recibirá solamente dos parámetros:

- El objeto jqxHR (el objeto jQuery XMLHttpRequest) utilizado en la petición
- Una cadena que contendrá una descripción del estado resultante de la petición, con uno de los siguientes valores:
    - success
    - notmodified
    - error
    - timeout
    - abort
    - parsererror

Opción beforeSend

Esta opción no controla una respuesta, pero podemos considerarla como parte del proceso de la ejecución de una petición. En el caso de esta opción, la función indicada como valor del mismo será llamada antes de que se realice la petición. El uso de esta función está justificado en dos casos:

- Modificar las cabeceras del objeto jqXHR que va a realizar la petición
- Poder cancelar la ejecución de la aplicación, devolviendo como resultado de la función el valor false.

Cabe destacar qu la función recibirá en su llamada dos argumentos: El objeto jqXHR y un objeto anónimo, con los detalles de la petición a realizar.

Opción jsonpCallback

La opción jsonpCallback será de utilidad para peticiones de dominio cruzado jsonp.

En este parámetro podremos indicar el nombre de una función o directamente una función, que será ejecutada al completarse una petición con dataType jsonp.

Si utilizamos una versión de jQuery anterior a la 1.5, entonces no podremos establecer una función como valor del parámetro, ya que esta funcionalidad se añadión en dicha versión de la librería.

A partir de la versión 1.5 de jQuery, las opciones success, error y complete podrán recibir como parámetro un array de funciones, que se ejecutarán una a una en el orden dispuesto en este array.

Otras opciones interesantes

Opción async

La opción async (que por defecto contendrá el valor booleano true) indicará si la petición será síncrona (si se establece a false) o asíncrona (si se establece a true).

El uso del modo asíncrono permite realizar varias peticiones al servidor de forma paralela, lo que permitirá que la carga de la información sea más rápida. Sin embargo, este método no nos asegura que las peticiones sean respondidas en el mismo orden en el que sean realizadas, ya que este funcionamiento solamente se obtendrá con peticiones síncronas.

En caso de que las peticiones se realicen en modo síncrono, las peticiones al servidor serán realizadas de una en una y se tardará más tiempo en cargar información. Las peticiones en modo síncrono bloquearán temporalmente el navegador, pausando la continuidad de ejecución hasta que la petición sea completada.

Opción cache

Por medio de la opción cache podremos forzar que las peticiones no se vean afectadas por el uso de la caché del navegador.

Este parámetro afectará a las peticiones GET y HEAD (este segundo tipo de peticiones es menos frecuente), añadiendo a la cadena de consulta el valor "_={timestamp}", siendo timestamp una marca de tiempo, consiguiendo así que la solicitud no sea procesada a partir de una petición ya cacheada.

Esta opción utilizará el valor por defecto true a excepción de para peticiones con el dataType establecido a script o jsonp

Opción contentType

La opción contentType permitirá indicar en la petición el tipo de información que se está enviando al servidor. Por defecto, el tipo de contentType enviado será "application/x-www-form-urlencoded; charset=UTF-8"

Opciones username y password

Estas dos opciones permiten indicar un nombre de usuario y una contraseña para realizar peticiones al servidor que requieran autenticación por HTTP. Para estos casos, podremos establecer el usuario y la contraseña que deberán proporcionarse al servidor cuando la petición se realice.

Opciones del método .ajax()

Establecer las opciones por decto con el método $.ajaxSetup()

El método .ajaxSetup() establecerá las opciones por defecto que serán utilizados de forma predeterminada en las futuras peticiones AJAX que se realicen. Este método aceptará como parámetro un objeto anónimo, cuyos atributos podrán ser cualquiera de las opciones disponibles para las peticiones AJAX.

$.ajaxSetup(objetoAnonimo)

Al establecer una configuración por defecto, ésta será utilizada por jQuery al realizar una petición AJAX, siempre y cuando a éstas no se les proporcione un objeto que indique las opciones particulares de la petición.

La configuración por defecto se mantendrá latente mientras que la página no sea completamente recargada, o hasta que sea sobreescrita con una nueva llamada al método .ajaxSetup().

La configuración aplicada con este método afectará tanto a las llamadas realizadas al método .ajax() como a todos aquellos que se basen en el mismo, como los métodos $.get() y $.post().

El uso de configuraciones por defecto puede provocar comportamientos indeseados (dada la poca transferencia del mismo) y puede tener efectos colaterales en plugins de terceros. Por estas razones, el uso de configuraciones por defecto está desaconsejado.

Veamos cómo afecta esta configuración en una sencilla llamada al método $.ajax():

Establecer la configuración por defecto

Vamos a establecer la configuración para las peticiones AJAX, estableciendo por defecto el método de envío GET y un tiempo máximo de espera de respuesta de 2 segundos

```
 $.ajaxSetup({
   type: "get"
   timeout: 2000 //en milisegundos, 2000 milisegundos son 2 segundos
});
```

Realizar la petición AJAX

Al establecer esta configuración, todas las peticiones AJAX que se realicen incluirán en sus parámetros finales esta configuración. Por ello, esta llamada al método $.ajax():

```
$.ajax({
  url: "target.php"
});
```

Se comporará como si hubiese sido realizada de este otro modo:

```
$.ajax({
  url: "test.html",
  type: "get"
  timeout: 2000 //en milisegundos, 2000 milisegundos son 2 segundos
});
```


Métodos abreviados para peticiones Ajax

Anteriormente hemos visto cómo utilizar el método .ajax() de jQuery para realizar peticiones AJAX a bajo nivel. Esta forma de realizar las peticiones es correctamente funcional, pero jQuery nos brinda otras posibilidades con las que optimizar nuestro código y conseguir los mismos resultados con muchas menos líneas de código.

En este momento es cuando entran en escena los métodos abreviados. La libreríajQ uery nos ofrece cinco métodos abreviados con los que realizar peticiones AJAX

- El método .post()
- El método .get()
- El método .getJSON()
- El método .getScript()
- El método .load()

Método .post()

El método .post() es una versión abreviada del método .ajax() de jQuery, en el que la petición al servidor se realizará por medio del tipo POST. Las peticiones realizadas por medio del método .post() equivaldrán a la siguiente petición por el método .ajax():

```
$.ajax({
  type: "POST",
  url: url,
  data: data,
  dataType: dataType,
  success: success
});
```


El método aceptará un máximo de tres parámetros en el siguiente orden:

1. URL a la que realizar la petición
2. Cadena de consulta, que contendrá la información que se enviará a la URL indicada en el primer parámetro, a través de una petición HTTP POST
3. Función controladora, con la que poder gestionar la respuesta devuelta por el servidor
4. dataType esperado por la función controladora

```
$.post(

  //destino
  "target.php",

  //parámetros de llamada
  { country: "spain" },

  //función controladora de la respuesta
  function( data ) {
    console.log( data.hour );
    console.log( data.minute );
  },

  //dataType de respuesta esperada por el servidor
  "json"
);
```

Este ejemplo muestra un uso básico del método .post(). En la llamada, se indica como URL de destino la página "target.php" y se proporciona un parámetro (que deberá ser procesada por la página de destino) de nombre "country" y de valor "spain". Como respuesta a la petición se espera un dato estructurado en formato JSON, que posea los atributos "hour" y "minute". La función controladora de la respuesta símplemente mostrará por consola los dos atributos del objeto devuelto por el servidor.

Para establecer otras opciones de llamada deberemos hacer uso del método .ajaxSetup(), indicando en el mismo las opciones adicionales a utilizar en la ejecución de la solicitud.

Las peticiones realizadas a través del método HTTP POST no se verá afectada por la caché. Por ello, los parámetros opcionales "cache" y "ifModified" disponibles en la configuración de peticiones con el método .ajaxSetup() no afectarán al comportamiento de este tipo de solicitud.

Envío de un formulario con el método $.post()

Una de las tareas más frecuentes en el uso de la tecnología AJAX es el envío de la información de un formulario a un servidor web, que procesará la información enviada (por ejemplo, registrándola en una base de datos) y nos devolverá un mensaje de respuesta, indicándonos el éxito o el fracaso en el mismo.

Vamos a ver cómo realizar esta tarea utilizando el método .post() de jQuery, analizando los detalles del proceso.

En cualquiera de los escenarios en los que nos encontremos dispondremos de un formulario, con un identificador, y un conjunto de campos con la información que se enviará en el mismo:

```
 <h2>FORMULARIO ENVIADO POR AJAX</h2>
<form id="ajaxform" action="destino.php" method="POST" >
  <input type="text" name="nombre" id="nombre" />
  <input type="text" name="email" id="email" />
  <input type="submit" name="enviar" value="Enviar" id="sbmenviar" />
  <div id="msg_status"></div>
</form>
```

Para este ejemplo, vamos a capturar el evento submit del formulario y reemplazar su funcionamiento estándar reemplazándolo por una llamada al método .post() con la información del formulario.

```
$(document).ready(
  $("ajaxform").on("submit", function(e){
    //cancelamos el comportamiento por defecto del evento
    e.preventDefault();

    //obtenemos el atributo "action" del evento
    //para utilizarlo como URL en el envío
    var urlDestino = $(this).attr("action")

    //capturamos los valores de los elementos del formulario
    var dataEnviar = $(this).serialize();

    //realizamos la llamada al método con 4 parámetros
    $.post(
      //1- la URL a la que realizar la petición
      urlDestino,
      //2- los datos a enviar
      dataEnviar,
      //3- la función controladora de la respuesta
      function(dataRespuesta){
        if(dataRespuesta.result==1){
          $("#msg_status").addClass("resultOk");
        }
        else{
          $("#msg_status").addClass("resultFail");
        }

        $("#msg_status").html(dataRespuesta.resultText);
      },
      //4- el tipo de dato esperado como respuesta
      "json"
    );
  });
);
```

Si observamos el código de la función de respuesta observaremos que ésta esperará dos posibles tipos de respuesta del servidor:

Esta respuesta se recibirá cuando el servidor realice correctamente la tarea que debe atender, por ejemplo almacenar la información enviada en una base de datos. Como respuesta, el servidor devolverá el siguiente código, representativo de un objeto JSON:

```
{
  "result": 1,
  "resultText": "Registro guardado correctamente"
}
```

La respuesta fallida será indicativa de que el servidor no ha podido cursar la solicitud por diversos motivos: imposibilidad técnica de realizar la tarea, incorrección de los datos enviados o cualquier otra situación que pudiera ocurrir.

En este caso, la respuesta del servidor podrá tener un código parecido al siguiente:

```
{
  "result": 0,
  "resultText": "La direcci&oacute;n de correo electr&oacute;nico indicada pertenece a un usuario ya registrad
o"
}
```

Hay que diferenciar este tipo de respuesta de un posible error de transferencia (problemas de conexión, timeout) o de interpretación de la información devuelta por el servidor (formato json incorrecto). Estos otros errores se podrán capturar utilizando las opciones disponibles para las peticiones AJAX, con el método .ajaxSetup()

Cómo se puede apreciar en el código, conseguimos modificar el funcionamiento por defecto del formulario mediante la captura del evento, reutilizamos la información establecida en el formulario - tanto para indicar la URL de destino como para los datos a enviar - y por último realizamos la llamada al método .post().

Método .get()

Otro método abreviado para realizar peticiones con AJAX es el método .get().

Este método tiene un funcionamiento similar al método .post() en cuanto a que este método preestablecerá el tipo de petición que se realizará al servidor, utilizando el tipo GET.

Las peticiones GET son frecuentemente utilizadas para realizar consultas al servidor, siendo desaconsejadas para operaciones de modificación o eliminación de información. Las peticiones realizadas con este método equivaldrán a la siguiente petición por el método .ajax():

```
$.ajax({
  type: "GET",
  url: url,
  data: data,
  dataType: dataType,
  success: success
});
```

Si comparamos esta equivalencia con la existente respecto al método .post() podremos observar que la única diferencia se centra en el tipo de petición HTTP, siendo en este caso de tipo GET en lugar de POST. Además, este método también coincide con el método .post() en los parámetros que podrá aceptar:

1. URL a la que realizar la petición
2. Cadena de consulta
3. Función controladora, con la que poder gestionar la respuesta devuelta por el servidor
4. dataType esperado por la función controladora

Estos parámetros serán opcionales. pudiendo omitirse si no es necesario para la solicitud:

```
$.get(

  //recurso del que obtener info.
  "contact_info.htm",

  //no indicamos ningún parámetro con datos de la petición
  { country: "spain" },

  //función controladora de la respuesta
  function( htmlResponse ) {
    $("#ajaxContent").html(htmlResponse);
  },

  //dataType de respuesta esperada por el servidor
  "html"
);
```

Cargar un listado de datos a partir de una petición $.get()

Una de las funcionalidades para las que está pensado el método .get() es obtener información de un servidor, para ser mostrada por la página que solicita esta información. Podemos encontrar este tipo de funcionalidad a diario mientras hacemos uso de internet: redes sociales, banca online y cientos de herramientas online que aprovechan las posibilidades que ofrece este tipo de tecnología.

Vamos a ver cómo realizar una petición a un servidor y, mediante este método, obtener datos para ser mostrados en un listado. En nuestro ejemplo vamos a suponer que nuestro servidor dispone de una URL a la que le podremos indicar una categoría y que, como respuesta a nuestra petición, devolverá un objeto JSON con los productos existenes en la misma.

HTML

La página encargada de realizar las peticiones dispondrá del siguiente código HTML:

```
<div id="catalogoDisplay">
 <h1>Cat&aacute;logo Online</h1>
 <div class="filtroData">
  <span>Seleccione la categor&iacute;a para filtrar</span>
  <select id="filtroCategoria">
     <option value="">-- Categor&iacute;a --</option>
     <option value="01">Ordenadores Sobremesa</option>
     <option value="02">Ordenadores Port&aacute;tiles</option>
     <option value="12">Almacenamiento</option>
     <option value="15">Monitores</option>
  </select>
 </div>
 <div class="displayData">
  <table id="tabDatos">
   <thead>
    <tr>
     <th>ID</th>
     <th>Descripci&oacute;n</th>
     <th>Precio</th>
     <th>Disponible</th>
    </tr>
   </thead>
   <tbody>
   </tbody>
   <tfoot>
    <tr>
     <td class="tdStatus" colspan="4"></td>
    </tr>
    <tr class="tplProduct" style="display:none;">
     <td class="displayId"></td>
     <td class="displayDescription"></td>
     <td class="displayPrice"></td>
     <td class="displayAvailable"></td>
    </tr>
   </tfoot>
  </table>
 </div>
</div>
```

Especificaciones

Así, las especificaciones de las que disponemos sobre la URL a la que debemos llamar es la siguiente:

URL

La URL a la que debemos realizar la petición es "list_articulos.php".

Parámetros

El parámetro que debemos indicar debe llamarse "idcategoria", y dispondremos de un elemento "select" en nuestra página con un listado de éstas.

Respuesta

La respuesta será en formato JSON. Esta respuesta podrá contener dos estructuras diferentes, dependiendo de que el parámetro proporcionado corresponda realmente a una categoría válida o no:

Si es una categoría válida

Si el parámetro indicado en la petición se corresponde con una categoría válida, la respuesta mostrará la siguiente estructura:


```
{
  "result": true,
  "message": "Se han localizado 3 productos en la categor&iacute;a",
  "data": [
    {
      "productId": 1297,
      "description": "Portable HDD 1TB",
      "price": 89.99,
      "available": true
    },
    {
      "productId": 1304,
      "description": "USB pen drive 16G Laser Pointer",
      "price": 8.99,
      "available": true
    },
    {
      "productId": 1717,
      "description": "External HDD 2TB",
      "price": 84.99,
      "available": false
    }
  ]
}
```


Si no es una categoría válida o no existen productos en la categoría

Si por el contrario el dato indicado como categoría no es válido o la categoría indicada no contiene ningún producto, la respuesta del servidor será la siguiente:


```
{
  "result": false,
  "message": "La categor&iacute;a indicada no es v&aacute;lida o no existe ning&uacute;n producto en ella"
}
```

Código jQuery

Finalmente, el código que controlará los cambios de categoría y actuará en consecuencia es el siguiente;

```
$(document).ready(function(){
  $("#filtroCategoria").on("change", function(e){
    var categoryId = $(this).val();

    //vaciamos el tbody de la tabla #tabDatos por si hay registros previos
    $("#tabDatos tbody").empty();

    //establecemos el mensaje de estado en el td .tdStatus
    $("#tabDatos .tdStatus").html("Cargando datos...");

    $.get(
      "list_articulos.php",
      { "idcategoria" : categoryId },
      function(jsonData){
        //mostramos el valor del mensaje devuelto por el servidor
        $("#tabDatos .tdStatus").html(jsonData.message);

        if(jsonData.result == true){
          //para cada uno de los datos devueltos

          $.each(jsonData.data, function(i, item) {
            //duplicamos la plantilla de producto

            $plantilla = $("#tabDatos .tplProduct").clone();

            //preparamos la plantilla, eliminando la clase y quitando el atributo style
            $plantilla.removeClass("tplProduct").removeAttr("style");

            //establecemos la info del producto
            $plantilla.find(".displayId").html( item.productId );
            $plantilla.find(".displayDescription").html( item.description );
            $plantilla.find(".displayPrice").html( item.price);
            if(item.available){
              $plantilla.find(".displayAvailable").html( "Producto Disponible" );
            }
            else{

              $plantilla.find(".displayAvailable").html( "Producto fuera de stock" );
            }

            $("#tabDatos tbody").append($plantilla);
            //otra forma de realizar esta acción sería $plantilla.appendTo($("#tabDatos tbody"));
          });
        }
      },
      "json"
    );
  });
});
```

Método .getJSON()

El método .getJSON() proporciona un atajo para la realización de peticiones AJAX que esperen como respuesta un dato de tipo JSON.

Este método podrá recibir hasta tres argumentos cuando hagamos una petición con él:

1. La URL de destino a la que realizar la petición AJAX
2. Los datos enviados en la petición, en formato de objeto anónimo
3. Una función de retorno, que será llamada y a la que se proporcionará un parámetro que contendrá un objeto JSON con los datos devueltos por el servidor en respuesta a la petición.

```
$.getJSON(url, data, function(jsonData){});
```

Este método es una forma resumida de realizar lapetición con el método .ajax() de jQuery del siguiente modo:

```
 $.ajax({
   type: "GET"
   dataType: "json",
   url: url,
   data: data,
   success: success
});
```

Como podrás observar, jQuery permite hacer la misma acción de diferentes formas. El uso de uno u otro métodos se convierte en una decisión personal, siempre y cuando el uso de los mismos sea el correcto. Es recomendable conocer todos los métodos disponibles ya que nos dará la libertad de decidir cual de todas las opciones utilizar.

Recuerda que una de las ventajas de jQuery es que con muy poco código podemos lograr grandes resultados y una de las formas de reducir la cantidad de código es utilizando este tipo de métodos.

Método .getScript()

El método .getScript() está horientado a la carga de código JavaScript desde un servidor por medio de una petición HTTP de tipo GET. Además, una vez el servidor responda a esta petición, el código devuelto por el mismo será ejecutado.

El código cargado se ejecutará en un contexto global, y éste podrá utilizar otras variables ya disponibles y/o métodos jQuery dado que la librería ya se encontrará cargada en la página. Además, el método también podrá configurarse para que, una vez completada la carga del nuevo script, se ejecute una función anónima en la que podremos hacer uso de las funciones, objetos, variables y cualquier otro recurso cargado por la petición.

Al hacer una llamada a este método, jQuery lanzará la petición como un alias del método .ajax() con la siguiente configuración:

```
 $.ajax({
   type: "GET",
   dataType: "script",
   url: url,
   success: success
});
```

El método aceptará dos parámetros:

1. La URL del archivo JavaScript a cargar en la página
2. Una función anónima que será ejecutada cuando el script sea cargado

La función de éxito se ejecutará cuando el script sea completamente cargado. Esto no implica que el código del recurso se haya ejecutado por completo

En nuestro servidor se encuentra el archivo js/funciones.js cuyo código fuente es el siguiente:

```
 function secondstominutes(secs)
{
   return ((arguments[1]=(Math.floor(secs/60)))<10?"0":"")+arguments[1]+":"+((arguments[2]=secs%60)<10?"0":"")
 + arguments[2];
}
```

Dentro de nuestra página, se ejecuta la siguiente llamada al método .getScript():

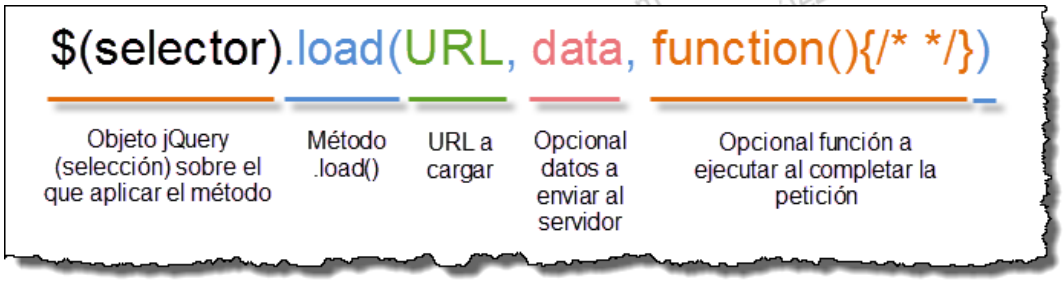
```
 $(document).ready(function(){
   $.getScript('js/funciones.js', function(){
      alert( secondstominutes(350) );
   });
});
```

Cuando el archivo js/funciones.js sea cargado dispondremos de la función "secondstominutes", que podremos invocar dentro de la función anónima indicada como parámetro al método.

Por defecto el método configurará la petición AJAX para evitar el uso de la caché del navegador. Es posible modificar este comportamiento indicando un valor diferente en la configuración por defecto por medio del método .ajaxSetup()

Método .load()

El último método para cargar información por medio de una solicitud AJAX es el método .load(). Es un método bastante sencillo cuya funcionalidad consiste en cargar la información a partir de una petición al servidor, estableciendo el dato devuelto por el mismo como contenido HTML de los elementos incluidos en la selección sobre la que se aplique el método.



Al llamar al méotodo .load() podremos indicar hasta tres parámetros:

1. La URL a cargar
2. Un objeto con los datos a enviar al servidor
3. Una función controladora, que se ejecutará cuando la petición sea completada.

A diferencia de los otros métodos anteriormente vistos, este método se aplicará sobre un objeto jQuery (selección), que utilizará como destinatarios en los que cargar la información devuelta por el servidor.

Cuando una llamada al método es correctamente procesada por el servidor, el método se encargará de establecer el contenido HTML en los elementos existentes en la selección. Generalmente las llamadas a este método se aplicarán sobre una selección por identificador de forma similar a la siguiente:

```
$("#artDesc").load( "html/articleDescription.html" );
```

Si la selección sobre la que se aplica el método no incluye ningún elemento (su atributo length tiene el valor 0) jQuery evitará realizar la solicitud y no se llevará a cabo ninguna petición al servidor.

URL a cargar

A diferencia de los demás métodos para realizar peticiones AJAX, el parámetro URL podrá tener una funcionalidad especial. Además de indicar la URL que el método deberá cargar, podremos especificar un marcador (selector por identificador), añadiéndolo al final de la URL y separándolo con un espacio del final de la misma.

```
$("#artDesc").load( "html/articleDescription.html #contenido" );
```

Este marcador hará que el método localice dentro del HTML devuelto por el servidor un elemento cuyo identificador coincida con el marcador. Así, si en éste existe algún elemento con ese identificador, solamente se establecerá el HTML de dicho elemento en los elementos de la selección, descartando el resto de HTML de la respuesta

Si la página devuelta contiene código JavaScript, el código será ejecutado anteriormente a establecer el HTML en los elementos destino. Esta funcionalidad no se dará cuando indiquemos un marcador en la URL.

Objeto con datos

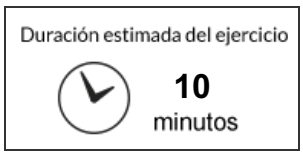
Otro aspecto a tener en cuenta cuando utilicemos el método load() viene dado por el tipo de dato que se le indique como parámetro con los datos a utilizar en la petición. Si este parámetro es un dato de tipo objeto, la petición al servidor se realizará por el método POST. En caso contrario, la petición al servidor se realizará por el método HTTP GET.

Función controladora de respuesta

Finalmente, el método podrá recibir una función anónima. Esta función se ejecutará cuando la petición se complete y el nuevo HTML se asigne a los objetos de la selección.

Ejercicios

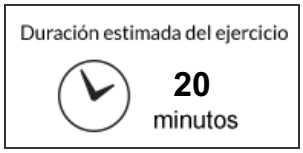
Ejercicio 1: Configuración de AJAX



Modifica por medio de métodos de jQuery la configuración por defecto para las peticiones AJAX de una página para que cumpla con las siguientes características:

1. Establece un tiempo máximo de petición de 20 segundos.
2. Uso de envíos por POST por defecto.
3. Indica que, el tipo de respuesta esperado sea por defecto JSON

Ejercicio 2: Envío de formulario por AJAX



Implementa el código jQuery necesario, sobre el fichero formulario_ajax.htm para que, en vez de realizar el envío estándar del formulario, utilice AJAX para realizar la petición. Dispondrá de dos botones de tipo "submit", que, cada uno de ellos, realizará las peticiones a:

- El "action" correcto.json
- El "action" erroneo.json

Los documentos correcto.json e incorrecto.json (incluidos en el archivo zip de descarga del ejercicio) contendrán las siguientes respuestas simuladas:

correcto.json

```
{
   "response": {
      "result": true,
      "message": "Usuario Registrado Correctamente"
   }
}
```

incorrecto.json

```
{
   "response": {
      "result": false,
      "message": "Errores en los datos enviados",
      "errorDetails": [
         {
            "fieldName": "email",
            "errorDesc": "El email indicado pertenece a otro usuario"
         },
         {
            "fieldName": "conocer",
            "errorDesc": "Indique la forma de conocimiento de nuestro sitio"
         }
      ]
   }
}
```


El mensaje del resultado debe ser mostrado en el div#divMessages

Lo necesario para comenzar

Descarga el archivo adjunto y extráelo en tu directorio de trabajo.

ingredientes-ejercicio-ajax.zip

Archivo comprimido con lo nece sario para realizar el ejercicio

Recursos

Enlaces de Interés

http://www.json-generator.com/

Herramienta para obtener objetos demostrativos en formato JSON

http://librosweb.es/ajax/capitulo_1.html

Introducción a AJAX

http://www.desarrolloweb.com/manuales/php-json.html

Trabajar con JSON desde PHP Manual para explicar el uso de JSON dentro de aplicaciones web realizadas con PHP para intercambiar información entre el cliente y el servidor.