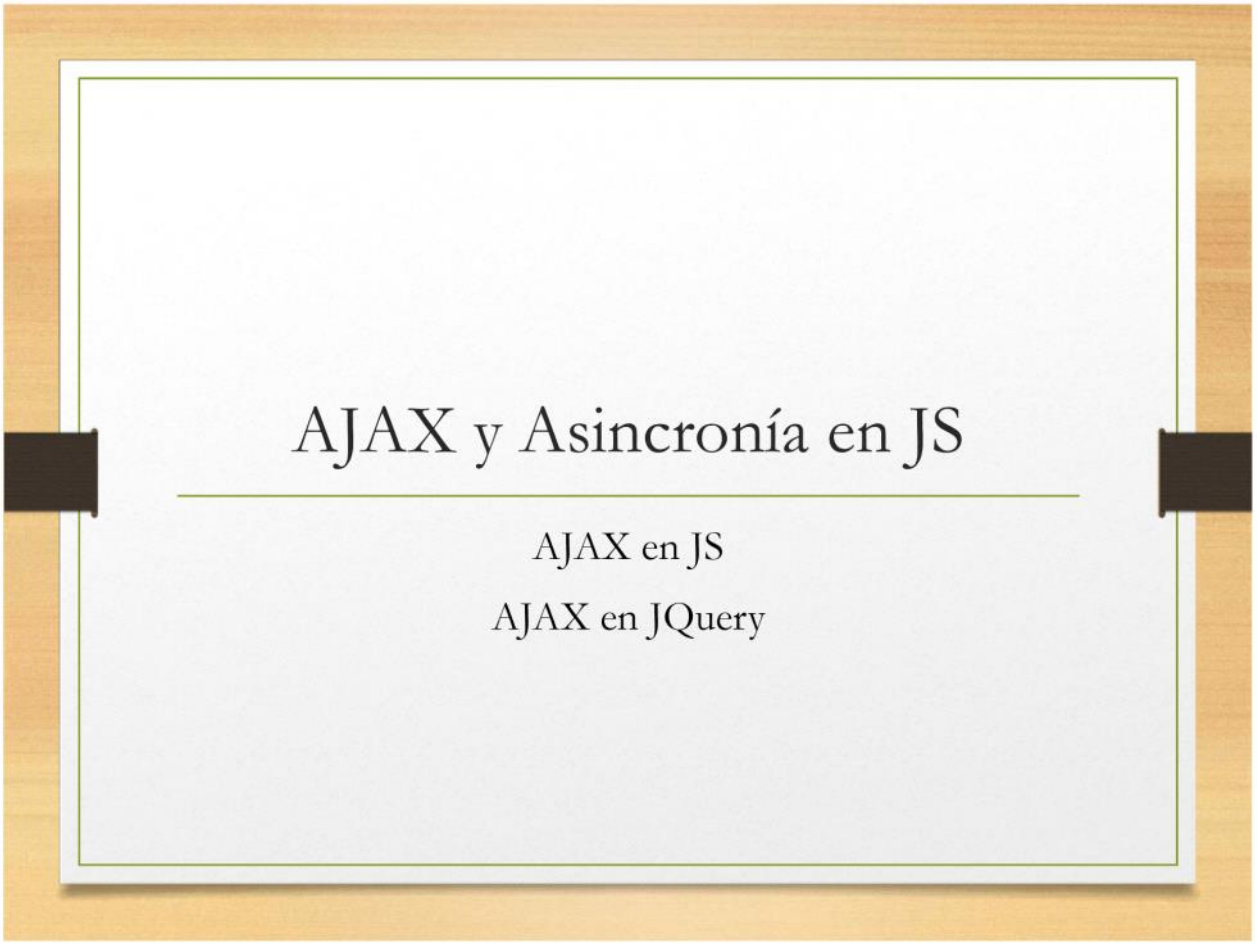
Asincronicidad: AJAX

domingo, 4 de junio de 2017 20:48

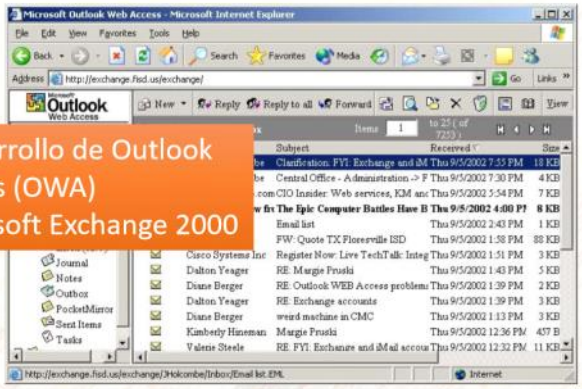


Ajax en JS

domingo, 4 de junio de 2017 20:48

AJAX: origen del XMLHTTP

Alex Hopmann



1998: Desarrollo de Outlook Web Access (OWA) para Microsoft Exchange 2000

tecnología que evitara tener que enviar continuamente los formularios con datos al servidor: **XMLHTTP**

Se incorpora en la librería MSXML de Internet Explorer 5

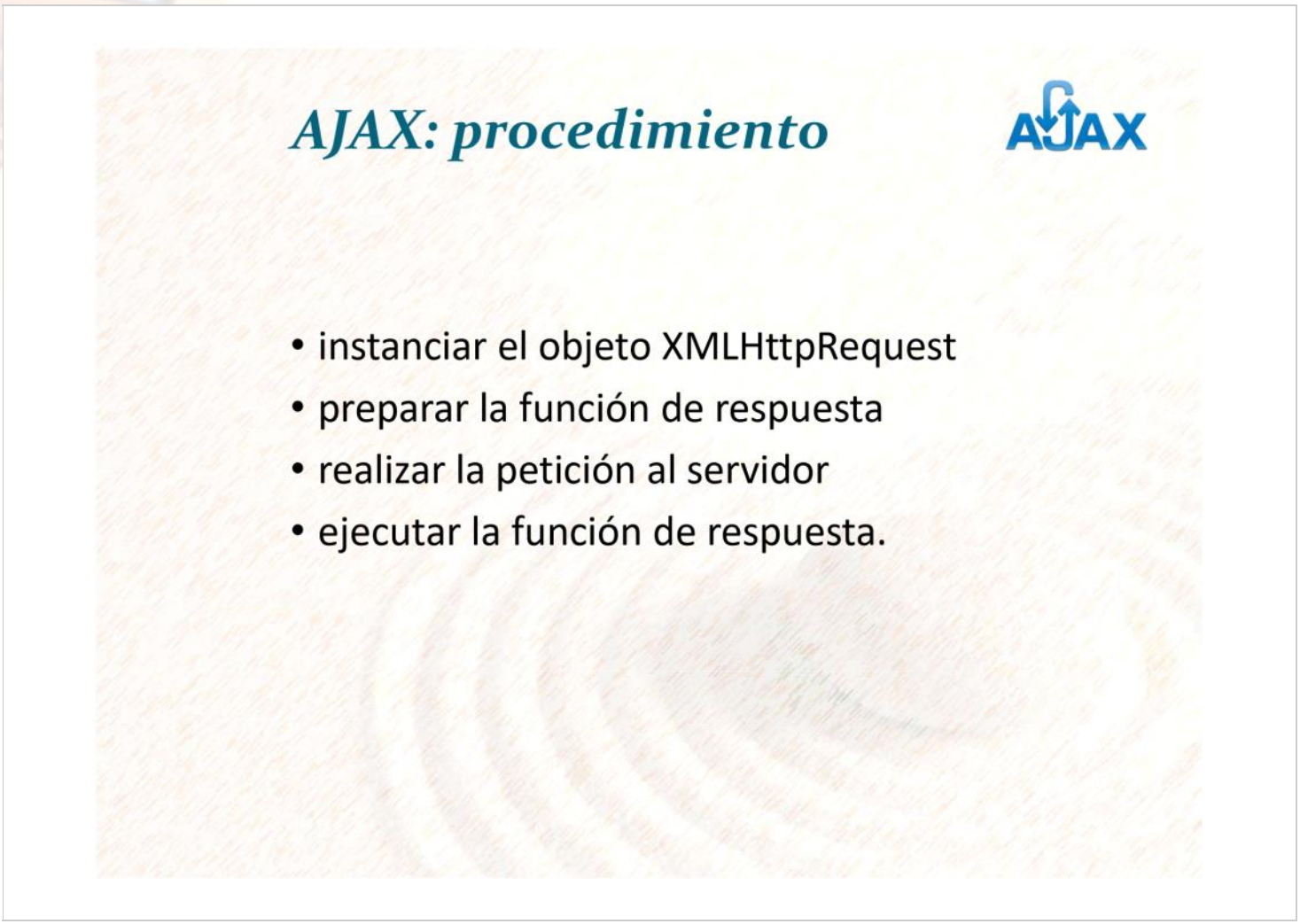
AJAX: Concepto

AJAX: Asynchronous JavaScript And XML (JavaScript asíncrono y XML)

asíncrona: los datos adicionales se solicitan al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página

el acceso a los datos se realiza mediante el **objeto XMLHttpRequest**, disponible en los navegadores actuales.

el contenido descargado del servidor puede estar o no formateado en **XML opcionalmente**. (se incluyo XML en la tecnología por motivos de estrategia empresarial). Alternativamente se puede usar HTML preformateado, texto plano, JSON, EBML...


AJAX: procedimiento
AJAX
• instanciar el objeto XMLHttpRequest
• preparar la función de respuesta
• realizar la petición al servidor
• ejecutar la función de respuesta.

Instanciación del objeto

Objeto XMLHttpRequest: permite realizar comunicaciones con el servidor en segundo plano, sin necesidad de recargar las páginas.

Su implementación **depende del navegador**:
es necesario emplear una discriminación sencilla entre MSIE 6 (o anteriores) y los otros navegadores (que siguen los estándares), según que propiedades están disponibles en el objeto window

```
if(window.XMLHttpRequest) {
     peticion_http = new XMLHttpRequest();
}
else if(window.ActiveXObject) { //
     peticion_http = new ActiveXObject("Microsoft.XMLHTTP");
}
```

Forma nativa del objeto en navegadores que siguen los estándares

Los navegadores obsoletos lo implementan como un objeto de tipo ActiveX

Función de respuesta

Una vez obtenida la instancia del objeto XMLHttpRequest, se prepara la función que se encarga de procesar la respuesta del servidor.

La propiedad **onreadystatechange** del objeto XMLHttpRequest permite indicar esta función directamente incluyendo su código mediante una función anónima o indicando una referencia a una función independiente.

```
peticion_http.onreadystatechange = muestraContenido;
```

En este caso se indica el nombre de la función que manejará el "evento" AJAX

Como siempre que se define un manejador de eventos, se indica su nombre sin paréntesis, ya que de otro modo se estaría ejecutando la función y almacenando el valor devuelto

Petición al servidor

Después de preparar la aplicación para la respuesta del servidor, **se realiza la petición HTTP al servidor**:

```
peticion_http.open('GET', 'http://localhost/prueba.txt', true);
peticion_http.send(null);
```

el tipo de petición más sencillo es una petición de tipo GET simple que no envía ningún parámetro al servidor.

- La petición HTTP **se crea** mediante el **método open(),** en el que se incluye el tipo de petición (GET), la URL solicitada (http://localhost/prueba.txt) y un tercer parámetro que vale true.
- Una vez creada la petición HTTP, **se envía** al servidor mediante el **método send()** con un parámetro de valor null.

Función de respuesta

Cuando se reciba la respuesta del servidor, se ejecutara la función definida previamente como manejadora del evento readystatechange.

```
function muestraContenido() {
   if(peticion_http.readyState == 4) {
      if(peticion_http.status == 200) {
        alert(peticion_http.responseText);}}}
```

- La función comprueba que se ha recibido la respuesta del servidor (mediante el valor de la propiedad readyState) y que dicha respuesta sea válida y correcta (comprobando si el código de estado HTTP devuelto es igual a 200).
- En caso correcto se muestra por pantalla el contenido de la respuesta del servidor (en este caso, el contenido del archivo solicitado) mediante la propiedad responseText.

Ejercicio (1a).

Hola mundo con AJAX.
Preparamos en el servidor un fichero holamundo.txt con el texto "Hola desde AJAX"

mediante AJAX se recoge el contenido del fichero, para luego mostrarlo mediante alert o incluirlo en la página web mediante innerHTML en el punto del DOM seleccionado

Objetivos. Conocer el uso básico de AJAX para descargar desde el servidor un fichero de texto, para luego mostrar su contenido.

Refactorización y otros enfoques

domingo, 4 de junio de 2017 20:56

Refactorización

```
const READY_STATE_UNINITIALIZED = 0;
const READY_STATE_LOADING = 1;
const READY_STATE_LOADED = 2;
const READY_STATE_INTERACTIVE = 3;
const READY_STATE_COMPLETE = 4;
var oPeticionHTTP;

function descargaArchivo() {
        cargaContenido('GET', 'holamundo.txt', muestraContenido);
}
```

Definimos como **constantes** los posibles valores de *readystatus*

Invocamos la función genérica

```
function cargaContenido(metodo, url, funcion){

    oPeticionHTTP = inicializa_xhr();
    // Obtener la instancia del objeto XMLHttpRequest

    if (oPeticionHTTP){
        oPeticionHTTP.onreadystatechange = funcion;
        oPeticionHTTP.open(metodo, url, true);
        oPeticionHTTP.send(null);

    }
}
```

función genérica de carga de contenidos, pasándole url, método y nombre de la función manejadora para que inicialice el objeto XMLHttpRequest

OOP en JavaScript: un objeto "AJAX"

Para realizar repetitivamente el proceso de solicitud de un recurso vía AJAX, se puede construir un objeto JavaScript con esa funcionalidad

realizar la carga de datos del servidor indicando

- el recurso solicitado
- la función encargada de procesar la respuesta:

Conceptos →

- objetos de JavaScript,
- funciones constructoras
- uso de prototype
- funciones anónimas
- JSON
- el objeto this
- control de errores try / catch

Ejercicio (1b).

Hola mundo con AJAX (mejorada).
Preparamos en el servidor un fichero holamundo.txt con el texto "Hola desde AJAX"

- Definimos como constantes los posibles valores de *readystatus*
- se refactoriza nuevamente (una función genérica de carga de contenidos, pasándole url, método y nombre de la función manejadora para que inicialice el objeto XMLHttpRequest) empleando un diseño OOP aplicando un patrón *singleton* (instancia única)

Objetivos. Conocer el uso básico de AJAX para descargar desde el servidor un fichero de texto, para luego mostrar su contenido.

El objeto XMLHttpRequest: propiedades

Propiedad	Descripción
readyState	Valor numérico (entero) que almacena el estado de la petición
responseText	El contenido de la respuesta del servidor en forma de cadena de texto
responseXML	El contenido de la respuesta del servidor en formato XML. El objeto devuelto se puede procesar como un objeto DOM
status	El código de estado HTTP devuelto por el servidor (200 para una respuesta correcta, 404 para "No encontrado", 500 para un error de servidor, etc.)
statusText	El código de estado HTTP devuelto por el servidor en forma de cadena de texto ("OK", "Not Found", "Internal Server Error", etc.)

El objeto XMLHttpRequest: eventos

Método	Descripción
onreadystatechange	evento que se invoca cada vez que se produce un cambio en el estado de la petición HTTP.

Como en cualquier otro evento, hay que definir la función responsable de manejarlo (*event handler*), bien como función anónima o, más habitualmente, como una referencia a una función JavaScript (que se indica mediante su nombre sin paréntesis)

La propiedad readyState: valores

Valor	Concepto	Descripción
0	No inicializado	objeto creado, pero no se ha invocado el método open
1	Cargando	objeto creado, pero no se ha invocado el método send
2	Cargado	se ha invocado el método send, pero el servidor aún no ha respondido
3	Interactivo	se han recibido algunos datos, aunque no se puede emplear la propiedad responseText
4	Completo	se han recibido todos los datos de la respuesta del servidor

En resumen

0: la petición no se ha inicializado
1: conexión con el servidor establecida
2: petición recibida
3: procesando petición
4: petición finalizada y la respuesta esta lista

El objeto XMLHttpRequest: métodos

Método	Descripción
abort()	Detiene la petición actual
getAllResponseHeaders()	Devuelve una cadena de texto con todas las cabeceras de la respuesta del servidor
getResponseHeader("cabecera")	Devuelve una cadena de texto con el contenido de la cabecera solicitada
open("metodo", "url")	Establece los parámetros de la petición que se realiza al servidor. Los parámetros necesarios son el método HTTP empleado y la URL destino (puede indicarse de forma absoluta o relativa)
send(contenido)	Realiza la petición HTTP al servidor. Si no se envían datos, el parámetro será NULL
setRequestHeader(“cabecera","valor")	Permite establecer cabeceras personalizadas en la petición HTTP. Se debe invocar el método open() antes que setRequestHeader()

El método XMLHttpRequest.open()

El método **open** acepta más parámetros de los que se muestran en la tabla; su esquema completo sería

```
open(string metodo, string URL
[,boolean asincrono, string usuario, string password])
```

- Por defecto, las peticiones realizadas son **asíncronas** (true). Si se indica un valor false al tercer parámetro, la petición se realiza de forma síncrona, esto es, se detiene la ejecución de la aplicación hasta que se recibe de forma completa la respuesta del servidor.
- Los últimos dos parámetros opcionales permiten indicar un **nombre de usuario** y una **contraseña** válidos para acceder al recurso solicitado.

Ejercicio (2a).

Lectura de un fichero con AJAX.
Preparamos en el servidor varios ficheros txt con diferentes textos
Creamos un breve formulario

- Input para indicar el nombre del fichero
- Boton para proceder a mostrar su contenido (via AJAX)

Recomendaciones

- Definimos como constantes los posibles valores de readystatus
- se crea una función genérica de carga de contenidos, pasándole url, método y nombre de la función manejadora para que inicialice el objeto XMLHttpRequest

Objetivos. Completar el uso básico de AJAX para descargar desde el servidor un fichero de texto, para luego mostrar su contenido.

Ejercicio (2b).

Lectura de un fichero con AJAX.
Completamos el ejercicio anterior capaz de seleccionar y leer varios ficheros txt en el servidor

- Añadimos información sobre el proceso de AJAX
- Añadimos la información suministrada por el servidor en las cabeceras de la respuesta

Seguimos aplicando las recomendaciones indicadas anteriormente

- constantes con los posibles valores de readystatus
- una función genérica de carga de contenidos,

Objetivos. Completar el uso básico de AJAX para descargar desde el servidor un fichero de texto, para luego mostrar su contenido.

Procesamiento de datos

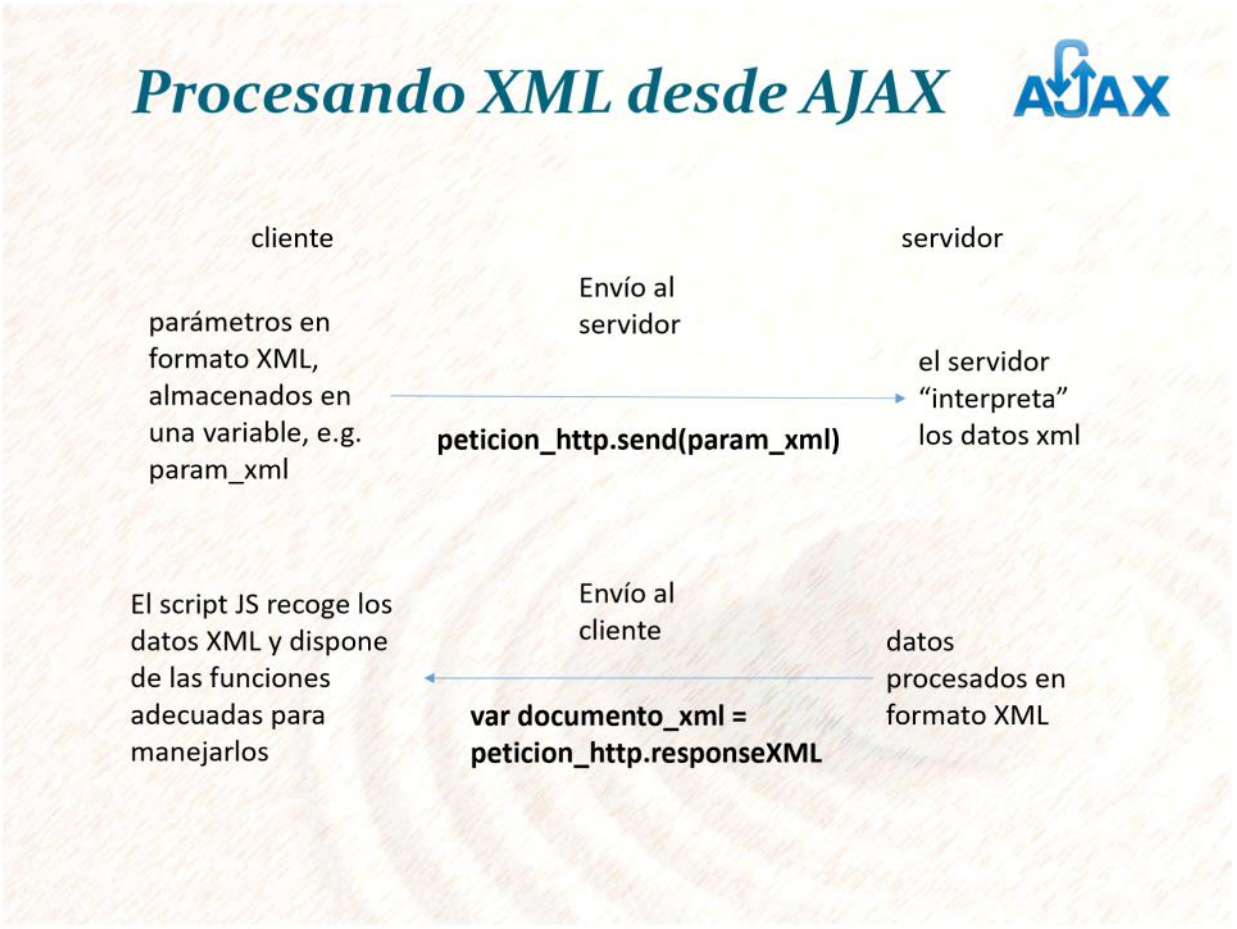
Pese a la X de su nombre, AJAX no esta ligado exclusivamente a XML

Formatos de datos habituales → HTML, XML, JSON

XML: muy flexible (nada predefinido); muy utilizado por sus similitudes con HTML (manipulado mediante las funciones del DOM); NO puede utilizarse de un dominio a otro

JSON: igualmente flexible; fácil de leer (para humanos y máquinas); SI permite el uso a través de dominios. Utiliza sus propias funciones de alto nivel: **JSON.stringify(object)** y **JSON.parse(string_JSON)**

Procesando XML desde AJAX



JSON: serialización

Serialización → transformación reversible de un tipo u objeto (en memoria) en un string equivalente

La serialización es un formato de intercambio de datos

- Almacenar datos en un fichero
- Enviar datos a través de una línea de comunicación
- Paso de parámetros en interfaces REST

En JavaScript (desde ECMAScript)

JSON.stringify(object)

objeto JavaScript se transforma a un string JSON

JSON.parse(string_JSON)

Un string JSON se transforma en el objeto original

Ejercicio (3a).

Lectura de un fichero JSON con AJAX.
Preparamos en el servidor un fichero JSON con diferentes textos
Utilizamos el formulario creado previamente

- Input para indicar el nombre del fichero
- Boton para proceder a mostrar su contenido (via AJAX) procesando correctamente el fichero JSON

Objetivos. Completar el uso básico de AJAX para intercambiar información con el servidor, sin necesidad de que en este se realice ningún procesamiento

Peticiones POST

domingo, 4 de junio de 2017 20:53

Peticiones HTTP: parámetros GET / POST

- Los parámetros GET o POST son siempre una cadena de pares: “nombre”:”valor”&“nombre”:”valor”….
- Existe una gran diferencia entre ambos
  - con el método POST los parámetros se envían en el cuerpo de la petición
    mediante el método GET los parámetros se concatenan a la URL
  - con el método GET existe un límite de 512 bytes en la cantidad de datos que se pueden enviar.
- Si se intentan enviar cadenas más largas el servidor devuelve un error con código 414 y mensaje Request-URI Too Long ("La URI de la petición es demasiado larga")

Al utilizar un objeto XMLHttpRequest no se generan automáticamente parámetros GET o POST, sino que la cadena que contiene los parámetros se debe construir manualmente.

Cadena de parámetros POST

XML.HttpRequest.setRequestHeader: genera la cabecera que define el tipo MIME de la solicitud. Para enviar datos POST, tipo formulario, es utiliza "application/x-www-form-urlencoded"
XML.HttpRequest.send(query_string): envía una cadena correctamente formateada, conocida como "query string". Lo habitual es crearla mediante una función.

```
if(peticion_http) {
    peticion_http.onreadystatechange = procesaRespuesta;

    peticion_http.open
        ("POST", "http://localhost/validaDatos.php", true);

    peticion_http.setRequestHeader
        ("Content-Type", "application/x-www-form-urlencoded");

    var query_string = crea_query_string();
    peticion_http.send(query_string);
}
```

Creación de la query string

La *query string* es la concatenación de las parejas nombre valor para cada uno de los elementos del formulario, separadas por &. Los valores se "formatean" según el procedimiento de las URL mediante la función encodeURIComponent()

```
function crea_query_string() {
    var fecha = document.getElementById("fecha_nacimiento");
    var cp = document.getElementById("codigo_postal");
    var telefono = document.getElementById("telefono");
    var query_string =
       "fecha_nacimiento=" + encodeURIComponent(fecha.value) +
       "&codigo_postal=" + encodeURIComponent(cp.value) +
       "&telefono=" + encodeURIComponent(telefono.value) +
       "&nocache=" + Math.random()
    return query_string;
}
```

Ejemplo completo

domingo, 4 de junio de 2017 21:01

Mini Proyecto.

Aplicación con AJAX y JSON. Creamos un sencillo catálogo (e.g. de libros) con una serie de imágenes miniaturas. Todas ellas ligadas a un procedimiento AJAX que accede a un fichero JSON específico para cada imagen, de forma que al pulsarla

- Se presentan los datos del elemento seleccionado
- Se actualiza su imagen en mayor tamaño

Objetivos. Completar el uso básico de AJAX para intercambiar información con el servidor, sin necesidad de que en este se realice ningún procesamiento

Detalles de la valoración

Funcionamiento - 5
(Eventos, Ajax, procesamiento JSON)

Diseño - 1
(HTML, CSS)

OOP - 1

Arquitectura del software - 1
(Distribución de clases - patrones)

Control de errores - 1

Buenas practicas - 1
(Comentarios, coherencia, indentación (sangrado)...)

API Fetch

viernes, 13 de abril de 2018 11:54

La **API `Fetch`** proporciona una interfaz JavaScript para acceder y manipular partes del flujo HTTP (HTTP pipeline), incluyendo peticiones y respuestas.

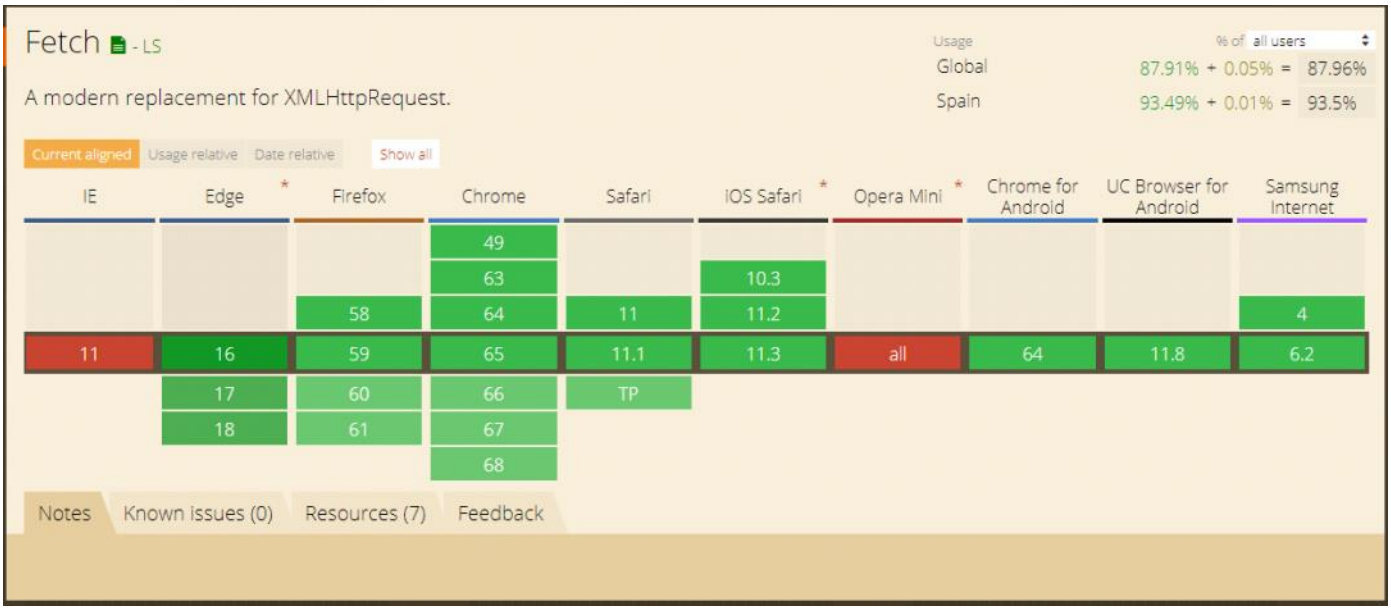
Proporciona un método global **`fetch()`** que supone un mecanismo fácil y lógico de obtener recursos por la red de forma asíncrona

El método **`fetch`**() devuelve una **promesa**, incluso en los casos de errores 404 o 500

Es una mejor alternativa al uso de XMLHttpRequest:
- puede ser empleada fácilmente por otras tecnologías como *Service Workers*.
- aporta un único lugar lógico en el que definir conceptos como CORS y extensiones a HTTP.

https://caniuse.com/#feat=fetch

Abril 2018



Método fetch()

viernes, 13 de abril de 2018 12:07

```
fetch('url', {configuración})

.then((response) => {
   return response.metod()
})
```

```
{ method: 'GET',
  headers: misCabeceras,
  body: data
  mode: 'cors',
  cache: 'default' }
```

```
arraybuffer()
blob()
formData()
json()
text()
```

La resolución de la promesa incluye un objeto *response* cuyo *body* puede ser procesado con una serie de métodos para distintos tipos de datos que en todos los casos devuelven una promesa.

```
.then((data) => {
   // transformación de los datos
})
```

https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/fetch

Cabeceras

```
myHeaders = new Headers({
  "Content-Type": "text/plain",
  "Content-Length": content.length.toString(),
  "X-Custom-Header": "ProcessThisImmediately",
});
```

POST / PUT

```
{ method: 'POST/PUT',
  headers: misCabeceras,
  body: JSON.stringyfy(data)
  mode: 'cors',
  cache: 'default' }
```

Ejemplo

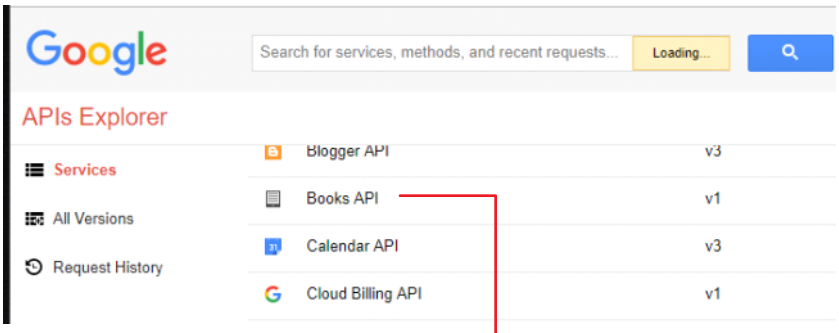
viernes, 13 de abril de 2018 12:37

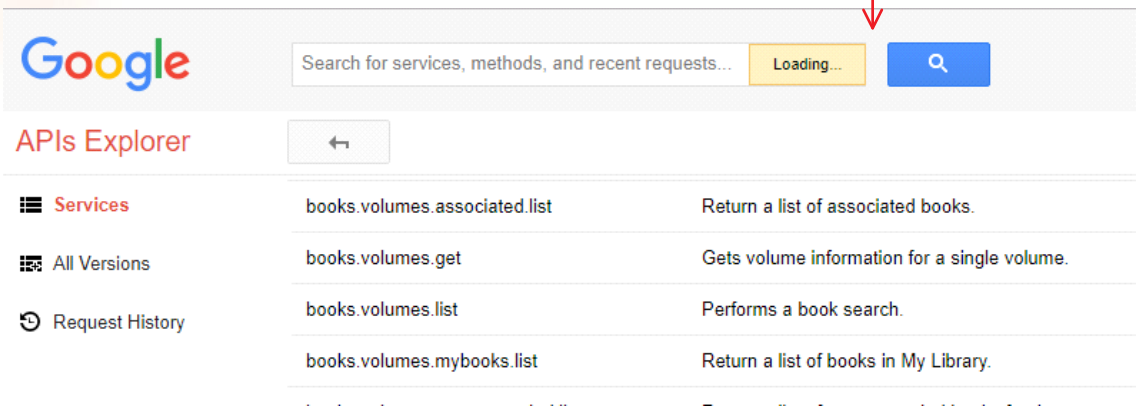
Ejercicio

viernes, 13 de abril de 2018 19:26

El API Books de Google

https://developers.google.com/apis-explorer/#p/





https://www.googleapis.com/books/v1/volumes?q=intitle:<clave>

q - Search for volumes that contain this text string. There are special keywords you can specify in the search terms to search in particular fields, such as:

- intitle: Returns results where the text following this keyword is found in the title.
- inauthor: Returns results where the text following this keyword is found in the author.
- inpublisher: Returns results where the text following this keyword is found in the publisher.
- subject: Returns results where the text following this keyword is listed in the category list of the volume.
- isbn: Returns results where the text following this keyword is the ISBN number.
- lccn: Returns results where the text following this keyword is the Library of Congress Control Number.
- oclc: Returns results where the text following this keyword is the Online Computer Library Center number.

https://developers.google.com/books/docs/v1/using

Promesas

sábado, 29 de julio de 2017 15:31

Una promesa representa el resultado eventual de una operación.
Se utiliza para especificar que se hará cuando esa eventual operación de un resultado de éxito o fracaso.

Promesas

Un objeto promesa representa un valor que todavía no esta disponible pero que lo estará en algún momento en el futuro

Permiten escribir código asíncrono de forma más similar a como se escribe el código síncrono:

La función asíncrona retorna inmediatamente y ese retorno se trata como un proxy cuyo valor se obtendrá en el futuro

El API de las promesas en Angular corresponde al **servicio $q**

la biblioteca Q desarrollada por **Kris Kowal**
https://github.com/kriskowal/q

Promesas: $q

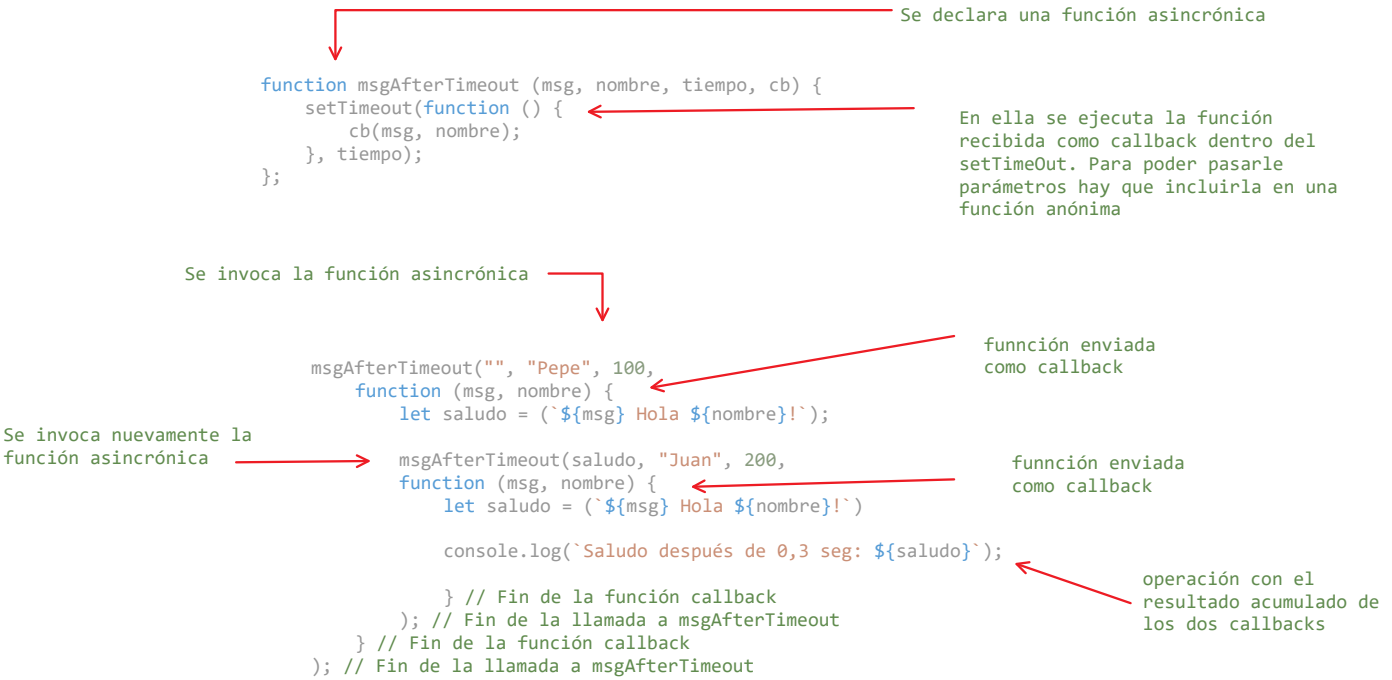
```
function getPromise()

    var deferred=$q.defer();          crea una promesa

        deferred.resolve()            resuelve la promesa en un sentido
        deferred.reject()             u otro al cabo del tiempo

    return deferred.promise           devuelve la promesa


var promise = getPromise();

    promise.then(successCallback,failureCallback,notifyCallback);
    promise.catch(errorCallback)
    promise.finally(callback)
```

promise.catch(errorCallback)

promise.finally(callback)

Callbacks anidados

sábado, 14 de octubre de 2017 13:52



Promesas en ES6

jueves, 10 de agosto de 2017 20:35

Implementación

`new Promise`

el objeto promesa recibe como parámetros dos funciones:

- La función "*resolve*": se ejecutará cuando queramos finalizar la promesa con éxito.
- La función "*reject*": se ejecutará cuando queramos finalizar una promesa informando de un caso de fracaso.

```
function hacerAlgoPromesa (){
    return new Promise (function (resolve, reject) {
        console.log ( 'hacer algo que ocupa un tiempo...');
        setTimeout (resolve(), 1000);
    })
}
```

En este caso la promesa siempre se resuelve correctamente; la función admite como parámetro los datos que la promesa deba retornar

Utilización

a la función que retorna el objeto promesa se le encadenan el método then con dos funciones como parámetros,

- la función que se ejecutará cuando la promesa haya finalizado con éxito.
- la función que se ejecutara cuando la promesa haya finalizado informando de un caso de fracaso.

```
hacerAlgoPromesa()
.then (
     function (){ console.log (' la promesa terminó.');},
    function (){ console.log (' la promesa fracasó.');}
    )
```

Alternativamente puede utilizarse el método catch para declarar la función que se ejecutara cuando la promesa haya finalizado informando de un caso de fracaso.

```
hacerAlgoPromesa()
.then (function (){ console.log (' la promesa terminó.');})
.catch(function (){ console.log (' la promesa fracasó.');})
```

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Promise

Ejemplo de promesas en ES6

sábado, 14 de octubre de 2017 14:43

Función que crea y devuelve un **objeto promesa**

```
function msgAfterTimeout (msg, nombre, tiempo) {

    return new Promise((resolve, reject) => {
        setTimeout(
            () => resolve(`${msg} Hola ${nombre}!`),
            tiempo)
        }
    )
}
```

En este caso, la promesa siempre se resuelve correctamente, creando un mensaje de saludo a un usuario

Utilización de las promesa, encadenando las llamadas a ellas

msg almacena el resultado del primer proceso asincrónico

msg almacena los sucesivos resultados de los procesos asincrónico

```
msgAfterTimeout("", "Pepe", 100)

.then((msg) =>
    msgAfterTimeout(msg, "Juan", 200))

.then((msg) => {
    console.log(`Saludo después de 0,3 seg: ${msg}`)
})
```

operamos finalmente con *msg*

```
MENSALES
Saludo despues de 0,3 seg:  Hola Pepe! Hola Juan!
PS D:\Desarrollo\Front_End_alce65\Angular\angular_4_2017\02_tecnologias\ES6>
```

Asinc / Await

domingo, 9 de septiembre de 2018 12:05

Definido en ES2017 (ES8)

La declaración de función async define una función asíncrona, la cual devuelve un objeto AsyncFunction.

```
function resolveAfter2Seconds() {
    return new Promise(resolve => {
        setTimeout(() => {
        resolve('resolved');
        }, 2000);
    });
}

async function asyncCall() {
    console.log('calling');
    var result = await resolveAfter2Seconds();
    console.log(result); // expected output: 'resolved'
}


asyncCall();
```