Desarrollo de aplicaciones multiplataforma

Elaborado por: José Antonio Sánchez

Módulo 3

Programación HTML5

Innova4B

SESIÓN 6

Innova4B

11.3. Web SQL

Web SQL **no** forma parte de la especificación de HTML5, pero es ampliamente utilizado para el desarrollo de aplicaciones web

http://www.w3.org/TR/webdatabase/

Se trata de una base de datos basada en SQL, más concretamente en SQLite

La utilización del API se resume en tres simples métodos:

- openDatabase: abrir (o crear y abrir) una base de datos en el navegador del cliente
- transaction: iniciar una transacción
- executeSql: ejecutar una sentencia SQL

Innova4B

11.3. Web SQL

El API de Web SQL es asíncrono, por lo que es necesario tener cuidado con el orden en el que se ejecutan las sentencias SQL

Sin embargo, las sentencias SQL se encolan y son ejecutadas en orden, por lo que podemos estar tranquilos en ese sentido: podemos crear tablas y tener la seguridad que van a ser creadas antes de acceder a los datos

11.3. Web SQL

openDatabase

Al abrir la base de datos por primera vez, ésta es creada automáticamente

```
var db = openDatabase('mydb', '1.0', 'My first database', 2 * 1024 * 1024);
```

Es necesario especificar el **número de versión** de base de datos con el que se desea trabajar, y si no especificamos correctamente éste número de versión, es posible obtener error del tipo INVALID_STATE_ERROR

El valor de retorno de esta función es un objeto que dispone de un método transaction, a través del cual vamos a ejecutar las sentencias SQL.

11.3. Web SQL

openDatabase

```
var db = openDatabase('mydb', '1.0', 'My first database', 2 * 1024 * 1024);
```

Los valores que pasamos a esta función son los siguientes:

- El nombre de la base de datos
- El número de versión de base de datos con el que deseamos trabajar
- Un texto descriptivo de la base de datos
- Tamaño estimado de la base de datos, en bytes

La última versión de la especificación incluye un quinto argumento en la función, pero no es soportado por muchos navegadores

11.3. Web SQL

transaction

La idea de utilizar transacciones, en lugar de ejecutar las sentencias directamente, es por la posibilidad de realizar rollback, es decir, volver al estado inicial

```
db.transaction(function (tx) {
    tx.executeSql('DROP TABLE foo');
    tx.executeSql('INSERT INTO foo (id, text) VALUES (1, "foobar")');
}, function (err) {
    alert(err.message);
});
```

En ocasiones se fuerza un rollback indicando una consulta errónea (dirty trick), ya que no existe un método “abort”

Innova4B

11.3. Web SQL

executeSql

El método executeSql es utilizado tanto para sentencias de escritura como de lectura

Incluye protección contra ataques de inyección SQL y proporciona llamadas a métodos (callback) para procesar los resultados devueltos por una consulta SQL

```
db.transaction(function (tx) {
    tx.executeSql(sqlStatement, arguments, callback, errorCallback);
});
```

Innova4B

11.3. Web SQL

executeSql

```
tx.executeSql(sqlStatement, arguments, callback, errorCallback);
```

- **sqlStatement**: indica la sentencia SQL a ejecutar: creación de tabla, insertar un registro, realizar una consulta, etc
- **arguments**: corresponde con un array de argumentos que pasamos a la sentencia SQL. El propio método se ocupa de prevenir inyecciones SQL
- **callback**: función a ejecutar cuando la transacción se ha realizado de manera correcta. Toma como parámetros la propia transacción y el resultado de la transacción
- **errorCallback**: función a ejecutar cuando la transacción se ha producido un error en la sentencia SQL. Toma como parámetros la propia transacción y el error producido

11.3. Web SQL

executeSql

```
db.transaction(function (tx) {
    tx.executeSql('SELECT * FROM foo WHERE id = ?',
        [5],
        function callback(tx, results) {
            var len = results.rows.length, i;
            for (i = 0; i < len; i++) {
            alert(results.rows.item(i).text);
            }
        },
        function errorCallback(tx, error) {
            alert(error.message);
        }
    );
});
```

Innova4B

11.3. Web SQL

executeSql

La función de callback recibe como argumentos la transacción (de nuevo) y un objeto que contiene los resultados

Este objeto contiene una propiedad rows, donde **rows.item(i)** contiene la representación de la fila concreta

Si nuestra tabla contiene un campo que se llama nombre, podemos acceder a dicho campo de la siguiente manera:

```
results.rows.item(i).nombre
```

11.3. Web SQL

Crear tablas

La primera tarea a realizar cuando trabajamos con una base de datos es crear las tablas necesarias para almacenar los datos

Este proceso se realiza a través de una sentencia SQL, dentro de una transacción:

```
db = openDatabase('tweetdb', '1.0', 'All my tweets', 2 * 1024 * 1024);
db.transaction(function (tx) {
    tx.executeSql('CREATE TABLE IF NOT EXISTS tweets(id, user, date, text)', [], getTweets);
});
```

Innova4B

11.3. Web SQL

Insertar datos

El siguiente paso es insertar los datos correspondientes en las tablas. En este ejemplo la función getTweets realizaría una llamada AJAX para obtener ciertos datos

```
function getTweets() {
        var tweets = $.ajax({...});
        $.each(tweets, function(tweet) {
                db.transaction(function (tx) { //CADA TWEET EN UNA TRANSACCION → Procesa todas
                        var time = (new Date(Date.parse(tweet.created_at))).getTime();
                        tx.executeSql('INSERT INTO tweets (id, user, date, text) VALUES (?, ?, ?, ?)',
                                [tweet.id, tweet.from_user, time / 1000, tweet.text]);
                });
        });
}
```

Innova4B

11.3. Web SQL

Obtener datos

Una vez que los datos se encuentran almacenados en la base de datos, sólo nos queda consultarlos y proporcionar la función que los procese

```
db.transaction(function (tx) {
    tx.executeSql('SELECT * FROM tweets WHERE date > ?', [time],
        function(tx, results) {
            var html = [], len = results.rows.length;
            for (var i = 0; i < len; i++) {
                html.push('<li>' + results.rows.item(i).text + '</li>');
            }
            tweetEl.innerHTML = html.join('');
        });
    });
}
```

Innova4B

11.4. Ejercicio práctico

Crear un objeto que encapsule una base de datos WebSQL, que nos permitar acceder a una base de datos para añadir, modificar, eliminar y obtener registros. Dicha base de datos va a almacenar tweets procedentes de Twitter, que tienen asociado el hashtag #html5. Los requisitos son los siguientes:

- Disponer de una tabla para almacenar los tweets. Los campos mínimos son: identificador del tweet, texto, usuario, y fecha de publicación
- Disponer de una tabla para almacenar los usuarios que publican los tweets. Esta tabla debe estar relacionada con la anterior. Los campos mínimos son: identificador del usuario, nombre e imagen
- Crear un método addTweet que dado un objeto que corresponde con un tweet, lo almacene en la base de datos. Almacenar el usuario en caso de que no exista, o relacionarlo con el tweet si existe

Innova4B

11.4. Ejercicio práctico

- Crear un método removeTweet que dado un identificador de tweet, lo elimine de la base de datos. Éste método debe devolver el tweet eliminado
- Crear un método updateTweet que dado un objeto que corresponde con un tweet, actualice los datos correspondientes al tweet en la base de datos
- Crear un método getTweets que dado un parámetro de fecha, me devuelva todos los tweets posteriores a esa fecha. Cada tweet debe incluir sus datos completos y el usuario que lo creó

Obtener los últimos 25 tweets que tienen como hashtag #html5 de la siguiente consulta a Twitter: http://search.twitter.com/search.json?q=%23html5&rpp=25&result_type="recent"

https://dev.twitter.com/docs/api/1/get/search

Innova4B

11.5. IndexedDB

IndexedDB **no** es una base de datos relacional, sino que se podría llamar un almacén de objetos

En la base de datos existirán almacenes y en su interior añadimos objetos

```
{
    id:21992,
    nombre: "Memoria RAM"
}
```

Innova4B

11.5. IndexedDB

Los pasos que seguiremos en IndexedDB serán:

- Abrir una base de datos indicando su nombre y la versión concreta
- Crear los almacenes de objetos, que es muy parecido a un archivador con índices, que nos permite encontrar de una manera muy rápida el objeto que buscamos
- Almacenar cualquier tipo de objeto con el índice que definamos
- No importa el tipo de objeto que almacenemos, ni tienen que tener las mismas propiedades

11.5. IndexedDB

Abrir la base de datos

Al abrir una base de datos por primera vez se creará, después podremos trabajar directamente con el objeto que nos devuelve

Al igual que en Web SQL, las peticiones a la base de datos se realizan de manera asíncrona

Previamente, necesitaremos hacer comprobaciones sobre el navegador

```
window.indexedDB = window.indexedDB || window.webkitIndexedDB || window.mozIndexedDB;
if ('webkitIndexedDB' in window) {
    window.IDBTransaction = window.webkitIDBTransaction;
    window.IDBKeyRange = window.webkitIDBKeyRange;
}
```

Innova4B

11.5. IndexedDB

Abrir la base de datos

Para abrir la base de datos únicamente necesitamos un nombre:

```
var request = indexedDB.open('videos');
request.onerror = function () {
        console.log('failed to open indexedDB');
};
request.onsuccess = function (event) {
        // handle version control
        // then create a new object store
};
```

Innova4B

11.5. IndexedDB

Abrir la base de datos

Ahora que la base de datos esta abierta (y asumiendo que no hay errores), se ejecutará el evento onsuccess

Antes de poder crear almacenes de objetos, tenemos que tener en cuenta los siguiente:

- Necesitamos un manejador para poder realizar transacciones de inserción y obtención de datos
- Hay que especificar la versión de la base de datos. Si no existe una versión definida, significa que la base de datos no está aún creada

11.5. IndexedDB

Abrir la base de datos

El método onsuccess recibe como parámetro un evento, dentro de este objeto, encontramos una propiedad llamada target, y dentro de ésta otra propiedad llamada result, que contiene el resultado de la operación

```
var db = null;
var request = window.indexedDB.open('videos');
request.onsuccess = function (event) {
        db = event.target.result;
        // Create a new object store
};
```

Innova4B

11.5. IndexedDB

Abrir la base de datos

```
request.onerror = function (event) {
        alert('Something failed: ' + event.target.message);
};
```

En IndexedDB, los errores que se producen escalan hasta el objeto request

Si un error ocurre en cualquier petición (por ejemplo una consulta de datos), se ejecutaría “request.onerror”

Innova4B

11.5. IndexedDB

Control de versiones

Como identificador de la versión podemos utilizar cualquier cadena, aunque lo lógico es seguir un patrón típico de software, como '0.1', '0.2'...

Al abrir la base de datos, debemos comprobar si la versión actual de la base de datos coincide con la última versión de la aplicación

En “setVersion” es el único lugar que podemos modificar la estructura de la base de datos: crear y eliminar almacenes de objetos e índices

11.5. IndexedDB

Control de versiones

```
var db = null, version = '0.1';
request.onsuccess = function (event) {
        db = event.target.result; // cache a copy of the database handle for the future
        if (version != db.version) { // handle version control
                // set the version to 0.1
                var verRequest = db.setVersion(version);
                verRequest.onsuccess = function (event) {
                        // now we're ready to create the object store!
                };
                verRequest.onerror = function () {
                        alert('unable to set the version :' + version);
                };
        }
};
```

Innova4B

11.5. IndexedDB

Control de versiones

```
request.onupgradeneeded = function(event) {
        var db = event.target.result;
        // Create object store
}
```

Si se necesita actualizar, crear o borrar la base de datos, también puede crearse el manejador para “onupgradeneeded”

Este evento se llama cuando se modifica la versión de la base de datos, o cuando se crea, por tanto es equivalente al caso anterior

Es en el único lugar que deberíamos poner las consultas que modifiquen la estructura de la base de datos

Innova4B

11.5. IndexedDB

Crear almacenes de objetos

Tanto la primera vez que creamos nuestra base de datos, como a la hora de actualizarla, debemos crear los almacenes de objetos correspondientes

```
var verRequest = db.setVersion(version);
verRequest.onsuccess = function (event) {
        var store = db.createObjectStore('blockbusters', {
                keyPath: 'title',
                autoIncrement: false
        });
        // Object store is ready!!
};
```

Innova4B

11.5. IndexedDB

Crear almacenes de objetos

Lo habitual es disponer de varios almacenes que puedan relacionarse entre ellos, así el método createObjectStore admite dos parámetros:

- name: nombre del almacén de objetos
- optionalParameters: define el índice de los objetos, a través del cual se van a realizar las búsquedas (**único**). Si no deseamos que este índice se incremente automáticamente al añadir nuevos objetos, también lo podemos indicar aquí (autoincrement: false)

Al añadir un objeto al almacén, es importante que disponga de la propiedad que se ha definido como índice (“title”), y que su valor sea único

11.5. IndexedDB

Crear almacenes de objetos

Es posible que deseemos añadir nuevos índices a nuestros objetos, con el fin de poder realizar búsquedas posteriormente

```
store.createIndex('director', 'director', { unique: false });
```

Hemos añadido un nuevo índice al almacén (“director”) y hemos indicado que el nombre de la propiedad del objeto es director (segundo argumento), a través del cual vamos a realizar las búsquedas

Como, varias películas pueden tener el mismo director, por lo que este valor no puede ser único

Innova4B

11.5. IndexedDB

Crear almacenes de objetos

De esta manera, podemos almacenar objetos y realizar búsquedas tanto por “title” (único) como por “director”

```
{
    title: "Belly Dance Bruce - Final Strike",
    date: (new Date).getTime(),
    director: "Bruce Awesome",
    length: 169, // in minutes
    rating: 10,
    cover: "/images/wobble.jpg"
}
```

Innova4B

11.5. IndexedDB

Añadir objetos al almacén

- add: añade un nuevo objeto al almacén. Es obligatorio que los nuevos datos no existan en el almacén, si no provocaría un ConstraintError
- put: actualiza el valor del objeto si existe, o lo añade si no existe en el almacén.

```
var video = {
        title: "Belly Dance Bruce - Final Strike",
        date: (new Date).getTime(),
        director: "Bruce Awesome",
        length: 169, // in minutes
        rating: 10,
        cover: "/images/wobble.jpg"
}
```

Innova4B

11.5. IndexedDB

Añadir objetos al almacén

```
var myIDBTransaction = window.IDBTransaction  || window.webkitIDBTransaction
        || { READ_WRITE: 'readwrite' };
var transaction = db.transaction(['blockbusters'], myIDBTransaction.READ_WRITE);
var store = transaction.objectStore('blockbusters');
// var request = store.add(video);
var request = store.put(video);
```

Crea una nueva transacción de lectura/escritura, sobre los almacenes indicados (en este caso sólo 'blockbusters')

Si no necesitamos que la transacción sea de escritura, se puede indicar con la propiedad myIDBTransaction.READ_ONLY

11.5. IndexedDB

Añadir objetos al almacén

```
var myIDBTransaction = window.IDBTransaction  || window.webkitIDBTransaction
        || { READ_WRITE: 'readwrite' };
var transaction = db.transaction(['blockbusters'], myIDBTransaction.READ_WRITE);
var store = transaction.objectStore('blockbusters');
// var request = store.add(video);
var request = store.put(video);
```

Obtiene el almacén de objetos sobre el que queremos realizar las operaciones, que debe ser uno de los indicados en la transacción

Con la referencia a este objeto, podemos ejecutar las operaciones de add, put, get, delete, etc

Innova4B

11.5. IndexedDB

Añadir objetos al almacén

```
var myIDBTransaction = window.IDBTransaction  || window.webkitIDBTransaction
        || { READ_WRITE: 'readwrite' };
var transaction = db.transaction(['blockbusters'], myIDBTransaction.READ_WRITE);
var store = transaction.objectStore('blockbusters');
// var request = store.add(video);
var request = store.put(video);
```

Inserta el objeto en el almacén

Si la transacción se ha realizado correctamente, se llamará al evento onsuccess, si no se llamará a onerror

Innova4B

11.5. IndexedDB

Obtener objetos del almacén

Se necesita una transacción, pero en este caso de sólo lectura

```
var myIDBTransaction = window.IDBTransaction || window.webkitIDBTransaction
        || { READ: 'read' };
var key = "Belly Dance Bruce - Final Strike";
var transaction = db.transaction(['blockbusters'], myIDBTransaction.READ);
var store = transaction.objectStore('blockbusters');
var request = store.get(key);
```

La variable “key” del método get, buscará el valor que contiene en la propiedad que definimos como “keyPath” al crear el almacén

Innova4B

11.5. IndexedDB

Obtener objetos del almacén

El método “get” produce el mismo resultado tanto si el objeto existe en el almacén como si no (devuelve undefined)

Es recomendable utilizar el método “openCursor()” para que si el objeto no existe, el valor del resultado sea *null*

Este método “openCursor” también permitirá obtener todos los objetos de un almacén, en lugar de un único objeto, pasando como parámetro un objeto de tipo IDBKeyRange

11.5. IndexedDB

Obtener objetos del almacén

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

- IDBKeyRange.upperBound(x) – Claves ≤ x
- IDBKeyRange.upperBound(x, true) – Claves < x
- IDBKeyRange.lowerBound(y) – Claves ≥ y
- IDBKeyRange.lowerBound(y, true) – Claves > y
- IDBKeyRange.bound(x, y) – Claves ≥ x && ≤ y
- IDBKeyRange.bound(x, y, true, true) – Claves > x && < y
- IDBKeyRange.bound(x, y, true, false) – Claves > x && ≤ y
- IDBKeyRange.bound(x, y, false, true) – Claves ≥ x && < y
- IDBKeyRange.only(z) – Clave = z

Innova4B

11.5. IndexedDB

Obtener objetos del almacén

```
var transaction = db.transaction(['blockbusters'], myIDBTransaction.READ);
var store = transaction.objectStore('blockbusters');
var data = [];
var request = store.openCursor();
// var singleKeyRange = IDBKeyRange.only("Belly Dance Bruce - Final Strike");
// var request = store.openCursor(singleKeyRange);
request.onsuccess = function (event) {
        var cursor = event.target.result;
        if (cursor) {
                data.push(cursor.value); // value is the stored object
                cursor.continue(); // get the next object
        } else {
                //Objects are in data[]
        }
};
```

Innova4B

11.5. IndexedDB

Eliminar objetos del almacén

```
var myIDBTransaction = window.IDBTransaction || window.webkitIDBTransaction
        || { READ_WRITE: 'readwrite' };

var transaction = db.transaction(['blockbusters'], myIDBTransaction.READ_WRITE);
var store = transaction.objectStore('blockbusters');
var request = store.delete(key);
```

Para eliminar todos los objetos de un almacén, podemos utilizar el método clear()

```
var request = store.clear();
```

Innova4B

11.6. Ejercicio práctico

Crear un objeto que encapsule una base de datos IndexedDB, que nos permita acceder a una base de datos para añadir, modificar, eliminar y obtener registros. Dicha base de datos va a almacenar una sencilla lista de tareas pendientes. Los requisitos son:

- Disponer de un almacén de tareas pendientes. Sus propiedades son: un identificador único que actúa como índice, el texto descriptivo, una propiedad que nos indique si la tarea está completada o no y la fecha/hora de creación
- Crear un método addTask que dado un objeto que corresponde con una tarea, lo almacene en la base de datos
- Crear un método removeTask que dado un identificador de una tarea, lo elimine de la base de datos. Éste método debe devolver la eliminada.
- Crear un método updateTask que dado un identificador de una tarea, actualice los datos correspondientes a la tarea en la base de datos
- Crear un método getTasks que dado un parámetro booleano completado, nos devuelva las tareas que se encuentran completadas o no

12. Web Workers

Los navegadores ejecutan las aplicaciones en un único thread

Si JavaScript está ejecutando una tarea muy pesada, que se traduce en tiempo de procesado, el rendimiento del navegador se ve afectado

Los Web workers se introdujeron con la idea de simplificar la ejecución de threads en el navegador

Un worker permite crear un entorno en el que un bloque de código JavaScript puede ejecutarse de manera paralela sin afectar al thread principal del navegador

12. Web Workers

Los Web workers se ejecutan en un subproceso aislado por tanto, es necesario que el código que ejecutan se encuentre en un archivo independiente

Lo primero que se tiene que hacer es crear un nuevo objeto Worker en la página principal

```
var worker = new Worker('task.js');
```

Si el archivo especificado existe, el navegador generará un nuevo subproceso de Worker que descargará el archivo JavaScript de forma asíncrona

El Worker no comenzará a ejecutarse hasta que el archivo se haya descargado completamente

12.1. API de transferencia de mensajes

Antes de comenzar a utilizar los Worker, es necesario conocer el protocolo de paso de mensajes

Este protocolo es utilizado en otras APIs como WebSocket y Server-Sent Event

El API de transferencia de mensajes es una manera muy simple de enviar cadenas de caracteres entre un origen (o un dominio) a un destino

Por ejemplo se puede utilizar para enviar información a una ventana abierta como popup, o a un iframe dentro de la página, aún cuando tiene como origen otro dominio

12.1. API de transferencia de mensajes

La comunicación entre un Worker y su página principal se realiza mediante un modelo de evento y el método postMessage()

En función del navegador o de la versión, postMessage() puede aceptar una cadena o un objeto JSON como argumento único

Siempre podemos utilizar los métodos JSON.stringify y JSON.parse para la transferencia de objetos entre el thread principal y los Worker

12.1. API de transferencia de mensajes

En la tarea principal:

```
var worker = new Worker('doWork.js');
worker.postMessage('Hello World');
```

En el Worker (doWork.js):

```
self.addEventListener('message', function(event) {
        self.postMessage(event.data);
}, false);
```

Al ejecutar postMessage() desde la página principal, el Worker es capaz de obtener este mensaje escuchando al evento “message” y puede acceder a los datos del mensaje a través de la propiedad “data” del evento

12.1. API de transferencia de mensajes

En la tarea principal:

```
var worker = new Worker('doWork.js');
worker.postMessage('Hello World');
```

En el Worker (doWork.js):

```
self.addEventListener('message', function(event) {
    self.postMessage(event.data);
}, false);
```

postMessage() también sirve para transferir datos de vuelta al thread principal

12.1. API de transferencia de mensajes

Los mensajes que se transfieren entre el origen y los Worker se **copian**, no se pasan por referencia.

El objeto se serializa al transferirlo al Worker y, posteriormente, se anula la serialización en la otra fase del proceso

El origen y el Worker **no comparten la misma instancia**, por lo que el resultado final es la creación de un duplicado en cada transferencia

12.1. API de transferencia de mensajes

En el documento principal

```
<button onclick="sayHI()">Say HI</button>
<button onclick="unknownCmd()">Send unknown command</button>
<button onclick="stop()">Stop worker</button>
<output id="result"></output>

<script>
	function sayHI() { worker.postMessage({'cmd': 'start', 'msg': 'Hi'});}
	function stop() { worker.postMessage({'cmd': 'stop', 'msg': 'Bye'});}
	function unknownCmd() { worker.postMessage({'cmd': 'foobard', 'msg':'???'});}

	var worker = new Worker('doWork.js');
	worker.addEventListener('message', function(e) {
		document.getElementById('result').textContent = e.data;
	}, false);
</script>
```

Innova4B

12.1. API de transferencia de mensajes

En el doWork.js

```
this.addEventListener('message', function(e) {
    var data = e.data;
    switch (data.cmd) {
        case 'start':
            this.postMessage('WORKER STARTED: '+data.msg);
            break;
        case 'stop':
            this.postMessage('WORKER STOPPED: '+data.msg+'. (buttons will no longer work)');
            this.close(); // Terminates the worker.
            break;
        default:
            this.postMessage('Unknown command: '+data.msg);
    };
}, false);
```

Innova4B

12.2. Uso de Web Workers

Un Worker es una manera ejecutar código JavaScript de manera paralela al proceso principal, sin interferir con el navegador

Al igual que con el resto de funcionalidades de HTML5, debemos comprobar su disponibilidad en el navegador en el que ejecutamos la aplicación:

```
if(Modernizr.webworkers) {
        alert('El explorador soporta Web workers');
} else {
        alert('El explorador NO soporta Web workers');
}
```

Crear nuevo Worker es muy sencillo. Tan sólo tenemos que crear una nueva instancia del objeto Worker, indicando como parámetro del constructor el fichero JavaScript que contiene el código que debe ejecutar el Worker.

12.2. Uso de Web Workers

Para crear un nuevo Worker tan sólo tenemos que crear una nueva instancia del objeto Worker, indicando como parámetro el fichero JavaScript que contiene el código que debe ejecutar el Worker

```
var worker = new Worker('my_worker.js');
```

La manera de comunicarnos con el nuevo Worker es a través API de transferencia de mensajes

```
worker.postMessage('Hello World');
```

Éste método únicamente acepta un parámetro, la cadena de texto a enviar al Worker

12.2. Uso de Web Workers

Por otra parte, la manera de recibir mensajes originados en el Worker es definiendo un escuchador para el evento message

```
worker.addEventListener('message', function(event) {
        alert(event.data);
}, false);
```

12.2. Uso de Web Workers

En el código del Worker

Dentro de un Worker necesitamos comunicarnos con el thread principal, tanto para recibir los datos de los mensajes como para nuevos datos de vuelta

Para ello, añadimos un escuchador para el evento “message”, y enviamos los datos de vuelta utilizando API de transferencia de mensages

```
this.addEventListener('message', function(e) {
    postMessage("I'm done!");
});
```

Innova4B

12.2. Uso de Web Workers

En el código del Worker

La visibilidad de un Worker es mucho más reducida, por ejemplo, la palabra reservada “this” no hace referencia al objeto window, sino al Worker en sí mismo

Los Workers tienen restringido el acceso a las siguientes funciones:

- DOM (no es seguro para el subproceso)
- Objeto window
- Objeto document
- Objeto parent

Innova4B

12.2. Uso de Web Workers

En el código del Worker

Debido al comportamiento de ejecución en paralelo de los Web workers, éstos solo pueden acceder a algunas funciones (según especificación):

- Enviar datos con postMessage y aceptar mensajes entrantes a través del evento onmessage
- close, para terminar con el Worker actual
- Realizar peticiones Ajax
- Utilizar las funciones de tiempo setTimeout()/clearTimeout() y setInterval()/clearInterval().
- Las siguientes funciones de JavaScript: eval, isNaN, escape, etc.
- WebSockets
- EventSource
- Bases de datos Web SQL, IndexedDB
- Web Workers

Innova4B

12.3. Subworkers

Los Workers tienen la capacidad de generar Workers secundarios (subtareas)

A la hora de utilizar estos Subworkers, y antes de poder devolver el resultado final al hilo principal, es necesario que todos los procesos hayan terminado

```
var pendingWorkers = 0, results = {};
onmessage = function (event) {
        var data = JSON.parse(event.data), worker = null;
        pendingWorkers = data.length;
        for (var i = 0; i < data.length; i++) {
                worker = new Worker('subworker.js');
                worker.postMessage(JSON.stringify(data[i]));
                worker.onmessage = storeResult(event);
        }
}
```

Innova4B

12.3. Subworkers

```
function storeResult(event) {
	var result = JSON.parse(event.data);

	pendingWorkers--;
	if (pendingWorkers <= 0) {
		postMessage(JSON.stringify(results));
	}
}
```

12.4. Gestión de errores

Si se produce un error mientras se ejecuta un Worker, se activa un evento error

La interfaz incluye tres propiedades útiles para descubrir la causa del error

- filename: el nombre de la secuencia de comandos del Worker que causó el error
- lineno: el número de línea donde se produjo el error
- message: una descripción significativa del error

Innova4B

12.4. Gestión de errores

```
function onError(error) {
        document.getElementById('error').textContent = [
                'ERROR: Line ', error.lineno, ' in ', error.filename, ': ',
                        error.message].join('');
}

function onMsg(event) {
        document.getElementById('result').textContent = event.data;
}

var worker = new Worker('workerWithError.js');
worker.addEventListener('message', onMsg, false);
worker.addEventListener('error', onError, false);
```

Innova4B

12.5. Seguridad

Debido a las restricciones de seguridad de Google Chrome, los Workers no se ejecutarán de forma local (por ejemplo, desde file://) en las últimas versiones del navegador

```
Uncaught Error: SECURITY_ERR: DOM Exception 18
```

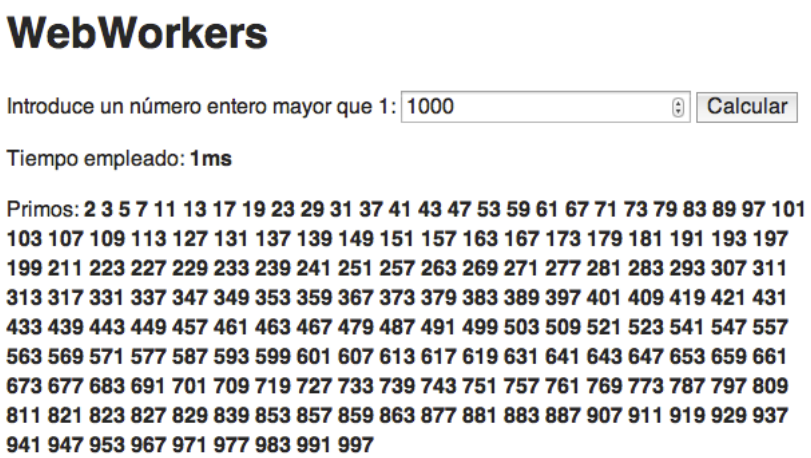
Para ejecutar tu aplicación desde el esquema file://, ejecuta Chrome con el conjunto de marcadores --allow-file-access-from-files

No se puede cargar una secuencia de comandos desde una URL **data:** o una URL **javascript:**. Tampoco, una página https: puede iniciar secuencias de comandos con una URL **http:**.

12.6. Ejercicio práctico

Crear un Web worker que dado un número entero, calcule todos los números primos comprendidos entre 1 y dicho número

Proporcionaremos a este Worker un número entero, y devolverá un array con todos los números primos encontrados. Mostrar el listado de números primos en el documento principal



Innova4B

13. XMLHttpRequest2

XMLHttpRequest2 introduce una gran cantidad de nuevas funciones que ponen fin a los problemas de nuestras aplicaciones web:

- Solicitudes de origen cruzado
- Eventos de progreso de subidas
- Compatibilidad con subida/bajada de datos binarios
- ...

Esto permite a AJAX trabajar en coordinación con muchas de las API HTML5, como API de FileSystem, el API de Web Audio y WebGL

Innova4B

13. XMLHttpRequest2

Recuperar archivos como “blob” binario era muy complicado con XHR

El truco que se ha documentado mucho implicaba anular el tipo mime con un conjunto de caracteres definido por el usuario:

```
var xhr = new XMLHttpRequest();
xhr.open('GET', '/path/to/image.png', true);
xhr.overrideMimeType('text/plain; charset=x-user-defined');
```

13. XMLHttpRequest2

```
xhr.onreadystatechange = function(e) {
        if (this.readyState == 4 && this.status == 200) {
                var binStr = this.responseText;
                for (var i = 0, len = binStr.length; i < len; ++i) {
                        var c = binStr.charCodeAt(i);
                        var byte = c & 0xff;  // byte at offset i
                }
        }
};
xhr.send();
```

Lo que se obtiene realmente en responseText es una cadena binaria que representa el archivo de imagen, no el tipo “blob”

Innova4B

13.1. Formato de respuesta

Las nuevas propiedades de XMLHttpRequest (responseType y response) permiten indicar al navegador el formato en el que queremos que nos devuelva los datos

- xhr.responseType: **antes** de enviar una solicitud, establece el tipo de respuesta a *text, arraybuffer, blob o document*. La respuesta predeterminada es *text*
- xhr.response: **después** de una solicitud correcta, la propiedad response de xhr contendrá los datos solicitados como DOMString, ArrayBuffer, Blob o Document (en función del valor establecido en responseType)

Con esto, podemos recuperar la imagen como ArrayBuffer en lugar de como una cadena

Innova4B

13.1. Formato de respuesta

```
BlobBuilder = window.MozBlobBuilder || window.WebKitBlobBuilder || window.BlobBuilder;

var xhr = new XMLHttpRequest();
xhr.open('GET', '/path/to/image.png', true);
xhr.responseType = 'arraybuffer';

xhr.onload = function(e) {
        if (this.status == 200) {
                var bb = new BlobBuilder();
                bb.append(this.response); // Note: not xhr.responseText
                var blob = bb.getBlob('image/png');
                // more code
        }
};
xhr.send();
```

Innova4B

13.1. Formato de respuesta

```
window.URL = window.URL || window.webkitURL;  // Take care of vendor prefixes.

var xhr = new XMLHttpRequest();
xhr.open('GET', '/path/to/image.png', true);
xhr.responseType = 'blob'; // Use Blob instead of ArrayBuffer

xhr.onload = function(e) {
        if (this.status == 200) {
                var blob = this.response;
                var img = document.createElement('img');
                img.onload = function(e) {
                        window.URL.revokeObjectURL(img.src); // Clean up after yourself
                };
                img.src = window.URL.createObjectURL(blob);
                document.body.appendChild(img);
                // more code
        }
};
xhr.send();
```

Innova4B

13.2. Envío de datos

XMLHttpRequest limitaba a enviar datos DOMString o Document (XML)

Se ha rediseñado el método “send()” para aceptar todos estos tipos:

- DOMString
- Document
- FormData
- Blob
- File
- ArrayBuffer

13.2. Envío de datos

Envío de la cadena de datos: xhr.send(domstring)

```
sendText('test string');
function sendTextNew(txt) {
        var xhr = new XMLHttpRequest();
        xhr.open('POST', '/server', true);
        xhr.responseType = 'text';
        xhr.onload = function(e) {
                if (this.status == 200) {
                        console.log(this.response);
                }
        };
        xhr.send(txt);
}
```

Innova4B

13.2. Envío de datos

Envío de formularios: xhr.send(formdata)

```
function sendForm() {
        var formData = new FormData();
        formData.append('username', 'johndoe');
        formData.append('id', 123456);

        var xhr = new XMLHttpRequest();
        xhr.open('POST', '/server', true);
        xhr.onload = function(e) { ... };
        xhr.send(formData);
}
```

Innova4B

13.2. Envío de datos

Envío de formularios: xhr.send(formdata)

Los objetos FormData se pueden inicializar a partir de un elemento HTMLFormElement de la página

```
<form id="myform" name="myform" action="/server">
    <input type="text" name="username" value="johndoe">
    <input type="number" name="id" value="123456">
    <input type="submit" onclick="return sendForm(this.form);">
</form>
```

Innova4B

13.2. Envío de datos

Envío de formularios: xhr.send(formdata)

```
function sendForm(form) {
        var formData = new FormData(form);
        formData.append('secret_token', '1234567890');
        var xhr = new XMLHttpRequest();
        xhr.open('POST', form.action, true);
        xhr.onload = function(e) { ... };
        xhr.send(formData);
        return false; // Prevent page from submitting.
}
```

Innova4B

13.2. Envío de datos

Envío de formularios: xhr.send(formdata)

FormData también puede incluir subidas de archivos. Simplemente se añade el archivo/s y el navegador construirá una solicitud multipart/form-data cuando se ejecute send()

```
function uploadFiles(url, files) {
    var formData = new FormData();
    for (var i = 0, file; file = files[i]; ++i) {
        formData.append(file.name, file);
    }
    var xhr = new XMLHttpRequest();
    xhr.open('POST', url, true);
    xhr.onload = function(e) { ... };
    xhr.send(formData);  // multipart/form-data automatic
}
```

Innova4B

13.2. Envío de datos

Envío de formularios: xhr.send(formdata)

Podremos llamar a la función de subida cuando se añada un archivo

```
document.querySelector('input[type="file"]').addEventListener('change', function(e) {
        uploadFiles('/server', this.files);
}, false);
```

Innova4B

13.2. Envío de datos

Envío de archivos o blob: xhr.send(blobOrFile)

Se puede crear un nuevo blob desde cero con el API BlobBuilder y se sube al servidor

Además también se puede configurar un controlador para informar al usuario sobre el progreso de la subida:

```
// Take care of vendor prefixes. BlobBuilder = window.MozBlobBuilder ||
                window.WebKitBlobBuilder || window.BlobBuilder;

var bb = new BlobBuilder();
bb.append('hello world');
upload(bb.getBlob('text/plain'));
```

Innova4B

13.2. Envío de datos

Envío de archivos o blob: xhr.send(blobOrFile)

```
function upload(blobOrFile) {
	var xhr = new XMLHttpRequest();
	xhr.open('POST', '/server', true);
	xhr.onload = function(e) { ... };

	var progressBar = document.querySelector('progress'); // Listen to the upload progress
	xhr.upload.onprogress = function(e) {
		if (e.lengthComputable) {
			progressBar.value = (e.loaded / e.total) * 100;
			progressBar.textContent = progressBar.value; // Unsupported browsers
		}
	};
	xhr.send(blobOrFile);
}
```

Innova4B

13.2. Envío de datos

Envío de fragmento de bytes: xhr.send(arraybuffer)

```
function sendArrayBuffer() {
        var xhr = new XMLHttpRequest();
        xhr.open('POST', '/server', true);
        xhr.onload = function(e) { ... };

        var uInt8Array = new Uint8Array([1, 2, 3]);

        xhr.send(uInt8Array.buffer);
}
```

Innova4B

13.3. Ejemplo. Descargar y guardar archivos en el sistema de archivos

Supongamos que tienes una galería de imágenes y quieres recuperar un grupo de imágenes para, a continuación, guardarlas localmente con el sistema de archivos HTML5

Una forma de conseguir esto sería solicitar imágenes como ArrayBuffer, crear un Blob a partir de los datos y escribir el blob con FileWriter:

```
window.requestFileSystem  = window.requestFileSystem ||
        window.webkitRequestFileSystem;

function onError(e) {
        console.log('Error', e);
}
```

Innova4B

13.3. Ejemplo. Descargar y guardar archivos en el sistema de archivos

```
var xhr = new XMLHttpRequest();
xhr.open('GET', '/path/to/image.png', true);
xhr.responseType = 'arraybuffer';
xhr.onload = function(e) {
        window.requestFileSystem(TEMPORARY, 1024 * 1024, function(fs) {
                fs.root.getFile('image.png', {create: true}, function(fileEntry) {
                        fileEntry.createWriter(function(writer) {
                                writer.onwrite = function(e) { ... };
                                writer.onerror = function(e) { ... };
                                var bb = new BlobBuilder();
                                bb.append(xhr.response);
                                writer.write(bb.getBlob('image/png'));
                        }, onError);
                }, onError);
        }, onError);
};
xhr.send();
```

Innova4B

13.3. Ejemplo. Dividir un archivo y subir cada fragmento

Con las API de archivo, podemos minimizar el trabajo necesario para subir un archivo de gran tamaño

La técnica es dividir el archivo que se va a subir en varios fragmentos, crear un XHR para cada parte y unir los fragmentos en el servidor

```
window.BlobBuilder = window.MozBlobBuilder || window.WebKitBlobBuilder ||
        window.BlobBuilder;

function upload(blobOrFile) {
    var xhr = new XMLHttpRequest();
    xhr.open('POST', '/server', true);
    xhr.onload = function(e) { ... };
    xhr.send(blobOrFile);
}
```

Innova4B

13.3. Ejemplo. Dividir un archivo y subir cada fragmento

```
document.querySelector('input[type="file"]').addEventListener('change', function(e) {
        var blob = this.files[0];
        const BYTES_PER_CHUNK = 1024 * 1024; // 1MB chunk sizes.
        const SIZE = blob.size;
        var start = 0;
        var end = BYTES_PER_CHUNK;
        while(start < SIZE) {
                if ('mozSlice' in blob) var chunk = blob.mozSlice(start, end);
                else var chunk = blob.webkitSlice(start, end);
                upload(chunk);
                start = end;
                end = start + BYTES_PER_CHUNK;
        }
}, false);
```

Innova4B