FileSystem

ALMACENAMIENTO EN EL LADO DEL CLIENTE

Óscar Vega Herrera
José Carlos Arjona Torres.

¿Qué es FileSystem?

- Es un conjunto de métodos y funcionalidades para crear, leer y modificar directorios y archivos en el navegador.

- Existe la API de FileSystem para permitir usar esas funcionalidades.

Ventajas e Inconvenientes

- Cómo ventaja, podemos indicar que usa un sistema de archivos muy familiarizado con el sistemas de archivos que usamos normalmente en local. Por ejemplo el uso de archivos, directorios, etc.
- Cómo desventaja, se puede mencionar que es una tecnología soportada sólo por el navegador de Google Chrome, y que en breve se dejará de usar.

Funcionamiento

- Solicitud de un sistema de archivos.
- Solicitud de espacio de almacenamiento.
- Operaciones con archivos.
- Operaciones con directorios.

Se debe ejecutar en un Servidor Web.

Solicitud de un sistema de archivos y de espacio de almacenamiento.

```
window.onload = function() {
    //  Para que reconozca la función requestFileSystem, en caso contrario, el intérprete
    //  de JavaScript no lo conoce y provoca error.
    window.requestFileSystem = window.requestFileSystem || window.webkitRequestFileSystem;

      //  Solicitud de espacio de almacenamiento.
    window.webkitStorageInfo.requestQuota(window.TEMPORARY, 1024*1024, function(grantedBytes) {

        //  En caso de que si se pueda solicitar espacio de almacenamiento.

        //  Solicitud de un sistema de archivos.
        window.requestFileSystem(window.TEMPORARY, 1024*1024, function(fs) {
            //  En caso afirmativo.
            console.log(`El sistema de archivos ${fs.name} ha sido abierto`);
        }, function(e) {
            //  En caso de que no se pueda solicitar.
            console.log(`Error durante la operación. Código de error: ${e.code}`);
        });

    }, function(e) {
        //  Error en caso que no se pueda solicitar espacio de almacenamiento.
        console.log('Error: ' + e.code);
    })

}
```

Solicitud de un sistema de archivos y de espacio de almacenamiento.

type: Indica si el almacenamiento de archivos debe ser permanente. Se usan los valores window.TEMPORARY (los datos se pueden eliminar a elección del navegador, por ejemplo si necesita más espacio), window.PERSISTENT (los datos no se pueden borrar, deberá autorizarlo el usuario o la aplicación).

size: Indica el espacio (en bytes) del almacenamiento que se requiere.

función: Función que se ejecuta en caso de afirmación. Su argumento es un objeto de FileSystem.

función: Función que se ejecuta en caso de error. Su argumento es un objeto de FileError.

Crear un archivo vacío.

```
function crearArchivo(nombre) {
    //  Se solicita el sistema de archivos.
    window.requestFileSystem(window.TEMPORARY, 1024*1024, function(fs) {

        //  Crea un archivo vacio
        fs.root.getFile(nombre, {create: true, exclusive: true}, function(fileEntry) {

            //  En caso de que se cree el archivo
            console.log(`Se ha creado el archivo ${fileEntry.name}`);
        }, function() {

            //  En caso que se produzca un error al intentar crear el archivo.
            //  Como que ya exista un archivo con el mismo nombre.
            console.log('Error, no se ha creado el archivo correctamente.');
        })

    });
}
```

Crear un archivo vacío.

fs.root.getFile() → Se debe ejecutar una vez que se haya solicitado el sistema de archivos. Pertenece a la instancia de FileSystem. fs en la instancia de FileSystem.

- Nombre del archivo: Nombre del archivo que se va a crear. Se puede especificar ruta.
- Objeto: Comportamiento que debe tener la función si el archivo no existe. En el caso de **create: true , exclusive: true** (hace que se cree el archivo si no existe y que genere un error en caso de que ya exista).
- Función anónima: Se ejecuta si se ha creado el archivo correctamente. Se le pasa como argumento una instancia de del archivo creado, donde se puede acceder a sus propiedades como “name”, que el nombre del archivo.
- Error: Función anónima que se ejecuta en caso de que se produzca un error al intentar crear el archivo.

Escribir en un archivo.

Se escribe contenido en el fichero vacío.

```
function crearArchivo(nombre) {
    //  Se solicita el sistema de archivos.
    window.requestFileSystem(window.TEMPORARY, 1024*1024, function(fs) {

        //  Crea un archivo vacio.
        fs.root.getFile(nombre, {create: true}, function(fileEntry) {

            //  Escribir contenido en el fichero.
            fileEntry.createWriter(function (fileWriter) {
                //  Se produce cuando se guardar contenido en el fichero.
                fileWriter.onwriteend = function(e) {
                    console.log('Se ha escrito en el fichero');
                }

                //  Se produce cuando se genera un error al guardar contenido.
                fileWriter.onerror = function(e) {
                    console.log(`Error al escribir el contenido ${e.toString()}`);
                }

                //  Escribir los datos de texto en el archivo.
                var bb = new Blob(['Contenido'], {type: 'text/plain'});
                fileWriter.write(bb);
            });
        }, function() {

            //  En caso que se produzca un error al intentar crear el archivo.
            //  Como que ya exista un archivo con el mismo nombre.
            console.log('Error, no se ha creado el archivo correctamente.');
        })

    });
}
```

Escribir en un archivo.

createWriter() →De FileEntry, para obtener un objeto de FileWriter. Se describen manejadores de eventos para los casos en el que se produzca error al escribir o si se ha realizado bién la escritura.

Se le pasa como argumento una función anónima que se encarga de realizar operaciones con la instancia de FileWriter.

Se crea una instancia Blob y se le pasa el texto de contenido. Esa instancia se le pasa al método FileWriter.write() para que realice la operación de escritura.

Leer un archivo por su nombre

```
function leerArchivo(nombre) {
    window.requestFileSystem(window.TEMPORARY, 1024*1024, function(fs) {

        //  Permite recuperar un archivo que ha sido guardado. Pasándole su nombre
        fs.root.getFile(nombre, {}, function (fileEntry) {

            //  Hace referencia al fichero que se quiere recuperar.
            fileEntry.file(function (file) {

                var reader = new FileReader();

                //  Se crea un textarea y se añade a la página con el contenido
                //  del fichero.
                reader.onloadend = function(e) {
                    var txtArea = document.createElement('textarea');
                    txtArea.value = this.result;
                    document.body.appendChild(txtArea);
                };
                reader.readAsText(file);
            })
        }, function () {
            //  Se ejecuta en caso de que el archivo no exista.
            console.log('El archivo no existe');
        })
    })
}
```

Leer un archivo por su nombre.

fs.root.getFile() → Se usa el mismo método para crear un archivo vacío, pero está vez, al pasarle como comportamiento de la función un objeto vacío, se obtiene una referencia del fichero.

- Si el fichero a buscar existe, se añade al contenido de la página, como un textarea con el contenido del fichero en su interior.

- En caso de que no exista el fichero, se ejecutará una función anónima..

Eliminar un archivo.

```
function elminiarArchivo(nombre) {
    //  Solicitud de espacio de almacenamiento.
    window.requestFileSystem(window.TEMPORARY, 1024*2014, function(fs) {

        //  Eliminar un archivo pasándole su nombre
        fs.root.getFile(nombre, {create:false}, function(fileEntry) {

            //  Elimina el archivo.
            fileEntry.remove(function() {
                console.log('Fichero borrado');
            });
        }, function() {
            //  Se ejecuta en caso de que no exista el fichero que se quiere borrar.
            console.log('El archivo no existe');
        })
    })
}
```

Eliminar un archivo.

Elimina un archivo pasándole el nombre del archivo.

fileEntry hace referencia al fichero que se quiere eliminar.

fs.root.getFile(nombreFichero, {create:false}, functionSucces(fileEntry), functionError()) → fileEntry hace referencia al fichero que se quiere borrar. En caso que se elimine, se ejecuta la función functionSucces(), en caso de que haya errores al intentar eliminar el fichero, como que no exista el fichero, se ejecuta la función functionError().

fileEntry.remove(function()) → Elimina el fichero que hace referencia, en caso de éxito, se ejecuta la función.

Crear directorio.

```
function crearDirectorio(nombre) {
    window.requestFileSystem(window.TEMPORARY, 1024*1024, function(fs) {
        //  Crear el directorio
        fs.root.getDirectory(nombre, {create:true}, function(dirEntry) {
            //  dirEntry representa el directorio creado.
            console.log(dirEntry);
        }, function() {
            //  Se ejecuta en caso de que haya errores al crear el directorio.
            console.log('Error, no se puede crear el dirctorio');
        })
    })
}
```

getDirectory() → Permite leer y crear directorios. Pasándole el nombre del directorio, el objeto que representa el funcionamiento de la función. dirEntry represente el directorio creado. En caso de error se ejecuta la otra función anónima, se puede ejecutar al no pasarle el nombre del directorio.

Dirección de fuente de información.

https://www.html5rocks.com/es/tutorials/file/filesystem/