JAVASCRIPT FRONTEND AVANZADO

CLASE 6 : UPLOADS / DOWNLOADS

DESCARGA

Aunque XMLHttpRequest se usa con más frecuencia para enviar y recibir datos de texto, se puede usar para enviar y recibir contenido binario. Hay varios métodos bien probados para forzar la respuesta de un XMLHttpRequest en el envío de datos binarios. Estos implican la utilización del método overrideMimeType () en el objeto XMLHttpRequest y es una solución viable:

```
var xhr = new XMLHttpRequest();
xhr.open("GET", url);
// recupera datos sin procesar como una cadena binaria
xhr.overrideMimeType("text/plain;
charset=x-user-defined");
```

Sin embargo, hay técnicas más modernas disponibles, ya que el atributo responseType ahora es compatible con varios tipos de contenido adicionales, lo que hace que enviar y recibir datos binarios sea mucho más fácil [1].

responseType

La propiedad **responseType** del objeto XMLHttpRequest puede configurarse para cambiar el tipo de respuesta esperada del servidor. Los valores posibles son la cadena vacía (predeterminada), "arraybuffer", "blob", "document", "json" y "text". La propiedad de respuesta contendrá el cuerpo de entidad de acuerdo con responseType, como ArrayBuffer, Blob, Document, JSON o string. Esto es nulo si la solicitud no está completa o no fue exitosa [2].

BLOB API

Un objeto Blob representa un objeto tipo fichero de datos planos inmutables. Los Blobs representan datos que no necesariamente se encuentran en un formato nativo de JavaScript. La interfaz File se encuentra basada en un Blob, heredando y expandiendo la funcionalidad de un Blob para soportar archivos en el sistema del usuario [3].

Lamentablemente si bien los blobs nos sirven para almacenar los datos binarios que podemos pedir por AJAX, el mismo no lo podemos usar para posterior construcción del DOM ya que es un objeto que tiene en su interior información que no podemos procesar por lo cual podemos usar otra API para extraer su contenido, como por ejemplo un método slice(), la API de FileReader o la API de URL para crear una URL en memoria del contenido que guarda nuestro blob.

URL

La interfaz **URL** representa a un objeto que provee métodos estáticos para crear objetos URL. Esta API nos provee de propiedades y métodos para usarlos de muchas otras maneras pero nosotros vamos a estar interesados en su método createObjectURL [4].

createObjectURL

El método estático URL.createObjectURL () crea un DOMString que contiene una URL que representa el objeto dado en el parámetro. La duración de la URL está vinculada al documento en la ventana en la que se creó. El nuevo objeto URL representa el objeto de archivo especificado o el objeto Blob, File ó MediaSource [5].

Con esta URL generada por la API podemos empezar a construir DOM dinámicamente :

```
var xhr = new XMLHttpRequest();
xhr.open("GET", url);
xhr.responseType = "blob";
xhr.addEventListener("load",function(e){
   if (xhr.status == 200) {
       var url = URL.createObjectURL(xhr.response)
       console.log(url)
   }
})
```

PROGRESO

XMLHttpRequest proporciona la capacidad de escuchar varios eventos que pueden ocurrir mientras se procesa la solicitud. Esto incluye notificaciones de progreso periódicas, notificaciones de errores, etc.

El soporte para la supervisión de evento de progreso DOM de las transferencias XMLHttpRequest sigue la especificación para eventos de progreso: estos eventos implementan la interfaz ProgressEvent. La interfaz ProgressEvent representa eventos que miden el progreso de un proceso subyacente, como una solicitud HTTP (para un XMLHttpRequest, o la carga del recurso subyacente de un <img>, <audio>, <video>, <style> o <link>) [6].

Los eventos de este tipo , entre otras , cuentan con las siguientes propiedades :

- **lengthComputable :** Es un indicador booleano que indica si el trabajo total que se debe realizar y la cantidad de trabajo ya realizado por el proceso subyacente es calculable. En otras palabras, dice si el progreso es mensurable o no.

- **loaded :** Es un número sin signo que representa la cantidad de trabajo ya realizado por el proceso subyacente. La proporción de trabajo realizado se puede calcular con la propiedad y ProgressEvent.total. Al descargar un recurso usando HTTP, esto solo representa la parte del contenido en sí, no los encabezados y otros gastos generales.

- **total :** Es un número sin signo que representa la cantidad total de trabajo del proceso subyacente. Al descargar un recurso usando HTTP, esto solo representa el contenido mismo, no los encabezados y otros gastos generales.

```
//...
xhr.addEventListener("progress",function(e){
   if(e.lengthComputable){
       console.log(e.loaded,e.total)
   }
})
//...
```

SUBIDA

Los eventos de progreso existen para las transferencias de descarga y carga. Los eventos de descarga se activan en el objeto XMLHttpRequest, como se muestra en la muestra anterior. Los eventos de carga se disparan en el objeto XMLHttpRequest.upload el cual es una interfaz del objecto XMLHttpRequestUpload .

XMLHttpRequestUpload API

Un objeto XMLHttpRequestUpload define un conjunto de propiedades de registro de controlador de eventos para supervisar el progreso de una carga de cuerpo de solicitud HTTP. En navegadores que implementan la especificación XMLHttpRequest Nivel 2, cada objeto XMLHttpRequest tiene una propiedad de carga que hace referencia a un objeto de este tipo.

Para supervisar el progreso de la carga de la solicitud, simplemente establezca estas propiedades en las funciones apropiadas del controlador de eventos o llame a los métodos EventTarget.

Tenga en cuenta que los controladores de eventos de progreso de carga definidos aquí son exactamente los mismos que los controladores de eventos de progreso de descarga definidos en XMLHttpRequest, excepto que no hay una propiedad on-state statechange en este objeto [7].

DRAG & DROP API

Las interfaces de arrastrar y soltar HTML permiten a las aplicaciones web arrastrar y soltar archivos en una página web. Este documento describe cómo una aplicación puede aceptar uno o más archivos que se arrastran desde el administrador de archivos de la plataforma subyacente y se eliminan en una página web.

Los pasos principales para arrastrar y soltar son definir una zona de colocación (dropzone : es decir, un elemento de destino para la eliminación del archivo) y definir handlers de eventos para los eventos de drop y dragover [8].

Hay que tener en cuenta además que estos eventos tienen un comportamiento por defecto en el navegador, de la misma forma que vimos con las etiquetas <a> y <form>, es por este

motivo que debemos cancelarlos para poder trabajar libremente con los listeners de cada evento :

```
//index.html
<div id="dropzone">
   <p>Arrastre sus archivos aquí</p>
</div>
```

```
//index.js
var dropzone = document.querySelector("#dropzone")
dropzone.addEventListener("dragover", e=>{
   e.preventDefault()
   e.stopPropagation()
})
dropzone.addEventListener("drop", e=>{
   e.preventDefault()
   e.stopPropagation()
   console.log("Archivo soltado!")
})
```

Los objetos evento de los eventos de tipo drag y drop cuentan con una propiedad bastante particular :

dataTransfer

El objeto DataTransfer se usa para contener los datos que se arrastran durante una operación de arrastrar y soltar. Puede contener uno o más elementos de datos, cada uno de uno o más tipos de datos. Estos elementos van a estar presente dentro de su propiedad **files** [9]**.**

Esta propiedad contiene una lista de todos los archivos locales disponibles en la transferencia de datos. Si la operación de arrastre no implica arrastrar archivos, esta propiedad es una lista vacía.

El evento de drop se activa cuando el usuario suelta el archivo (s). En el siguiente gestor de colocación, si el navegador admite la interfaz DataTransferItemList, el método getAsFile () se utiliza para acceder a cada archivo; de lo contrario, la propiedad de archivos de la interfaz DataTransfer se usa para acceder a cada archivo.

FILE API

La interfaz de File proporciona información sobre archivos y permite que JavaScript en una página web acceda a su contenido.

Los objetos File generalmente se recuperan de un objeto FileList devuelto como resultado de que un usuario seleccione archivos usando el elemento <input>, desde un objeto DataTransfer de una operación de arrastrar y soltar, o desde la API mozGetAsFile () en un HTMLCanvasElement. En Gecko, el código con privilegios puede crear objetos File que representen cualquier archivo local sin interacción del usuario (para más información, consulte Notas de implementación).

Un objeto File es un tipo específico de Blob y se puede usar en cualquier contexto que pueda tener Blob. En particular, FileReader, URL.createObjectURL (), createImageBitmap () y XMLHttpRequest.send () aceptan Blobs y Archivos [10].

FORMDATA API

La interfaz FormData proporciona una forma de construir fácilmente un conjunto de pares clave / valor que representan campos de formulario y sus valores, que luego pueden enviarse fácilmente utilizando el método XMLHttpRequest.send (). Utiliza el mismo formato que un formulario usaría si el tipo de codificación se configura como "multipart / form-data".

Un objeto que implementa FormData se puede usar directamente en una estructura for ... of, en lugar de entries (): for (var p of myFormData) es equivalente a for (var p of myFormData.entries ()) [11].

Esta API además de poder construirse desde un formulario :

```
var formData = new FormData(form)
```

El cual es conveniente ya que cuando se especifica el “form” , el objeto FormData se completará con las claves / valores actuales del formulario utilizando la propiedad del nombre de cada elemento para las claves y su valor presentado para los valores (También codificará el contenido de entrada de archivos) puede obtener datos externos a través de su método append.

append

El método append () de la interfaz FormData agrega un nuevo valor a una clave existente dentro de un objeto FormData, o agrega la clave si aún no existe.

La diferencia entre FormData.set y append () es que si la clave especificada ya existe, FormData.set sobrescribirá todos los valores existentes con el nuevo, mientras que append () agrega el nuevo valor al final del conjunto existente de valores.

```
formData.append(name, value);
```


1. https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Using_XMLHttpRequest
2. https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/responseType
3. https://developer.mozilla.org/en-US/docs/Web/API/Blob
4. https://developer.mozilla.org/en-US/docs/Web/API/URL
5. https://developer.mozilla.org/en-US/docs/Web/API/URL/createObjectURL
6. https://developer.mozilla.org/en-US/docs/Web/API/ProgressEvent
7. https://www.safaribooksonline.com/library/view/javascript-the-definitive/9781449393854/rn02re315.html
8. https://developer.mozilla.org/en-US/docs/Web/API/HTML_Drag_and_Drop_API/File_drag_and_drop
9. https://developer.mozilla.org/en-US/docs/Web/API/DataTransfer
10. https://developer.mozilla.org/en-US/docs/Web/API/File
11. https://developer.mozilla.org/en-US/docs/Web/API/FormData