DESARROLLO DE APLICACIONES WEB FRONT END

Módulo 4

OTEC PLATAFORMA 5 CHILE

MÓDULO 4

CLASE 4

Introducción

Cabeceras HTTP

POST Formularios

Proceso de carga

Resumen

Introducción

Hoy terminaremos con la explicación de XMLHttpRequest, en la cual veremos conceptos claves para el envío de formularios.

Luego de esto, saltaremos a otros temas, los cuales son necesarios tener estos conceptos bien establecidos.

Para la próxima clase, trabajaremos con un entorno gráfico para estos métodos.

Cabeceras HTTP

XMLHttpRequest permite tanto enviar cabeceras personalizadas como leer cabeceras de la respuesta.

Existen 3 métodos para las cabeceras HTTP:

- **setRequestHeader(name, value)**

Asigna la cabecera de la solicitud con los valores name y value provistos.

Por ejemplo:

xhr.setRequestHeader('Content-Type', 'application/json');

LNo se pueden eliminar cabeceras

Otra peculiaridad de XMLHttpRequest es que no puede deshacer un setRequestHeader.

Una vez que una cabecera es asignada, ya está asignada.

Cabeceras HTTP

Llamadas adicionales agregan información a la cabecera, no la sobreescriben.

Por ejemplo:

```
xhr.setRequestHeader('X-Auth', '123');
xhr.setRequestHeader('X-Auth', '456');
// la cabecera será:
// X-Auth: 123, 456
```

- **getResponseHeader(name)**

Obtiene la cabecera de la respuesta con el name dado (excepto Set-Cookie y Set-Cookie2).

Por ejemplo:

```
xhr.getResponseHeader('Content-Type')
```

Cabeceras HTTP

- **getAllResponseHeaders()**

Devuelve todas las cabeceras de la respuesta, excepto por Set-Cookie y Set-Cookie2.

Las cabeceras se devuelven como una sola línea, ej.:

```
Cache-Control: max-age=31536000
Content-Length: 4260
Content-Type: image/png
Date: Sat, 08 Sep 2012 16:53:16 GMT
```

El salto de línea entre las cabeceras siempre es un "\r\n" (independiente del SO), así podemos dividirlas en cabeceras individuales. El separador entre el nombre y el valor siempre es dos puntos seguido de un espacio ": ". Eso quedó establecido en la especificación.

Así, si queremos obtener un objeto con pares nombre/valor, necesitamos tratarlas con un poco de JS.

Como esto (asumiendo que si dos cabeceras tienen el mismo nombre, entonces el último sobreescribe al primero):

```
let headers = xhr
 .getAllResponseHeaders()
 .split("\r\n")
 .reduce((result, current) => {
   let [name, value] =
current.split(": ");
   result[name] = value;
   return result;
 }, {});

// headers['Content-Type'] =
'image/png'
```

POST Formularios

Para hacer una solicitud POST, podemos utilizar el objeto FormData nativo.

La sintaxis:

```
let formData = new FormData([form]); // crea un objeto, opcionalmente se completa con un <form>
formData.append(name, value); // añade un campo
```

Lo creamos, opcionalmente lleno desde un formulario, append (agrega) más campos si se necesitan, y entonces:

1. `xhr.open('POST', ...)` **– se utiliza el método POST.**
2. `xhr.send(formData)` **para enviar el formulario al servidor.**

si nos gusta más JSON, entonces, un JSON.stringify y lo enviamos como un string.

Solo no te olvides de asignar la cabecera Content-Type: application/json, muchos frameworks del lado del servidor decodifican automáticamente JSON con este:

```
let xhr = new XMLHttpRequest();

let json = JSON.stringify({
 name: "Michael",
 surname: "Jackson",
});

xhr.open("POST", "/submit");
xhr.setRequestHeader("Content-type",
"application/json; charset=utf-8");

xhr.send(json);
```

Progreso de carga

El evento progress se dispara solo en la fase de descarga.

Esto es: si hacemos un POST de algo, XMLHttpRequest primero sube nuestros datos (el cuerpo de la respuesta), entonces descarga la respuesta.

Si estamos subiendo algo grande, entonces seguramente estaremos interesados en rastrear el progreso de nuestra carga. Pero xhr.onprogress no ayuda aquí.

Hay otro objeto, sin métodos, exclusivamente para rastrear los eventos de subida: xhr.upload.

Progreso de carga

Este genera eventos similares a xhr, pero xhr.upload se dispara solo en las subidas:

- **loadstart – carga iniciada.**
- **progress – se dispara periódicamente durante la subida.**
- **abort – carga abortada.**
- **error – error no HTTP.**
- **load – carga finalizada con éxito.**
- **timeout – carga caducada (si la propiedad timeout está asignada).**
- **loadend – carga finalizada con éxito o error.**

Ejemplos de manejadores

```
xhr.upload.onprogress = function
(event) {
 alert(`Uploaded ${event.loaded} of
${event.total} bytes`);
};

xhr.upload.onload = function () {
 alert(`Upload finished
successfully.`);
};

xhr.upload.onerror = function () {
 alert(`Error durante la carga:
${xhr.status}`);
};
```

En resumen

De hecho hay más eventos, la especificación moderna los lista:

- **loadstart – la solicitud ha empezado.**
- **progress – un paquete de datos de la respuesta ha llegado, el cuerpo completo de la respuesta al momento está en response.**
- **abort – la solicitud ha sido cancelada por la llamada de xhr.abort().**
- **error – un error de conexión ha ocurrido, ej. nombre de dominio incorrecto. No pasa con errores HTTP como 404.**
- **load – la solicitud se ha completado satisfactoriamente.**
- **timeout – la solicitud fue cancelada debido a que caducó (solo pasa si fue configurado).**
- **loadend – se dispara después de load, error, timeout o abort.**

Los eventos error, abort, timeout, y load son mutuamente exclusivos. Solo uno de ellos puede pasar.

Los eventos más usados son la carga terminada (load), falla de carga (error), o podemos usar un solo manejador loadend y comprobar las propiedades del objeto solicitado xhr para ver qué ha pasado.

MÓDULO 4

GRACIAS

Aquí finaliza la clase n°1 del módulo 4

OTEC PLATAFORMA 5 CHILE