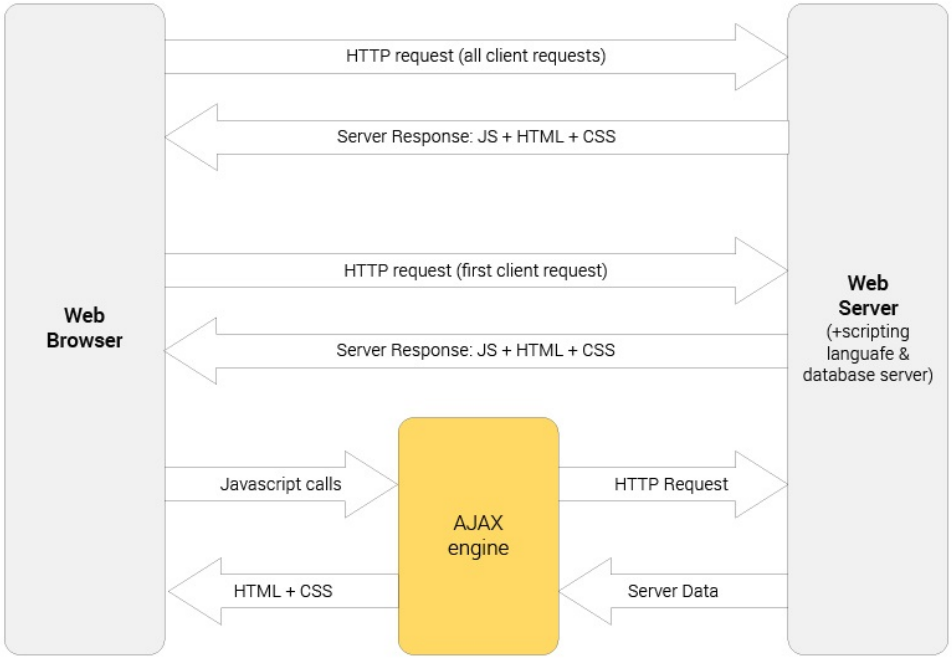
Asynchronous Javascript And XML (AJAX)

Noelia Barreira nbarreira@udc.es

November 10, 2021

AJAX

- Combinación de objetos/funciones del navegador para realizar peticiones de datos con Javascript y HTML DOM para mostrar dichos datos
- Permite actualizar páginas web de forma asíncrona sin recargar la página completa



AJAX application communication model

Cross-Origin HTTP Request (CORS)

- Recurso de un dominio A hace peticiones de otro recurso a un dominio B
- Habitual para
  - Ficheros de estilos CSS
  - Imágenes
  - Scripts Javascript
- Por motivos de seguridad, los navegadores no permiten realizar peticiones HTTP a otros dominios desde scripts

Peticiones AJAX

Legacy

- IE5, IE6

```
xhttp = new ActiveXObject("Microsoft.XMLHTTP")
```

- Chrome, IE7+, Firefox, Safari, Opera

```
xhttp = new XMLHttpRequest( );
```

  - Crear una petición

```
xhttp.open(method, url, async)
```

    method GET, POST, PUT, DELETE, ...
    async true/false

  - Enviar una petición

```
xhttp.send() // GET
xhttp.send(string) // POST
```

Peticiones AJAX

Legacy

- Procesar respuesta

```
xhttp.onreadystatechange = callback
```

callback función que se ejecutará cada vez que estado de la propiedad readyState

- Estados posibles de `readyState`

0 petición no inicializada
1 conexión establecida con el servidor
2 petición recibida
3 petición procesándose
4 petición terminada y respuesta preparada

- Otras propiedades del objeto `XMLHttpRequest`

status código de estado devuelto por el servidor tras procesar la petición (200, 400, 404, ...)
statusText texto asociado al código de estado

Peticiones AJAX

Legacy

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
  // this hace referencia a xhttp
  if (this.readyState == 4
  && this.status == 200) {
    dataAsString = this.responseText;
    // dataAsXML = this.responseXML;
  }
};
xhttp.open("GET", "http://server.com/api/example", true)
    ;
xhttp.send();
```

Peticiones AJAX

ECMAScript 6 (2015)

- Reemplazo de `XMLHttpRequest` por `fetch`
- Soportado en la mayor parte de los navegadores desde 2017
- No soportado en
  - Internet Explorer
  - Opera Mini
  - Blackberry browser

Fetch

```
Promise<Response> fetch(resource[, init])
```

Parámetros de entrada

- **resource** recurso que se solicita (url, objeto `Request`)
- **init** configuración de la petición
  - **method** tipo de petición (GET, POST, PUT, DELETE...)
  - **headers** cabeceras que se incluyen en la petición
  - **body** cuerpo del mensaje (`Blob`, `FormData`, `BufferSource`, `ReadableStream`...)
  - **mode** modo en el que se hará la petición (`cors`, `no-cors`, `same-origin`)
  - **credentials** credenciales para realizar la petición
  - **cache** modo de cache
  - **redirect** modo de redirección usado
  - ...

Fetch

```
Promise<Response> fetch(resource[, init])
```

Valores de retorno

- `Promise`
  - Es un proxy que permite asociar código para gestionar qué hacer si la función asíncrona termina de forma correcta o devuelve un código de error.
  - Dispone de tres métodos

    promise.then() se ejecuta cuando la petición asíncrona termina de forma exitosa

    promise.catch() se ejecuta cuando se produce una excepción

    promise.finally() se ejecuta al final, independientemente del resultado de la petición asíncrona

Fetch

```
Promise<Response> fetch(resource[, init])
```

Valores de retorno

- `Promise` que se resuelve en un objeto `Response`
- Propiedades de `Response`

ok	booleano que indica si la respuesta fue existosa
status	código de estado de la respuesta (200, 400, 404, ...)
statusText	mensaje correspondiente al código de estado
headers	cabeceras
type	tipo de la respuesta
url	de la respuesta
	...

Fetch

```
Promise<Response> fetch(resource[, init])
```

Valores de retorno

- `Promise` que se resuelve en un objeto `Response`
- Métodos de `Response`
  - Leen el stream de la respuesta y devuelven una `Promise` que se resuelve en otro objeto

arrayBuffer() devuelve una `Promise` a un `ArrayBuffer`
blob() devuelve una `Promise` a un `Blob`
formData() devuelve una `Promise` a un `FormData`
json() devuelve una `Promise` a un `JSON`
text() devuelve una `Promise` a un `USVString` (texto)

Ejemplos de uso

Petición GET con respuesta JSON

```
fetch("https://dog.ceo/api/breeds/image/random")
 .then(response => {
    if (response.ok) {
      // Returns a Promise to a JSON object
      return response.json();
    } else {
      throw Error('Error');
    }
  })
 .then(response =>
  // Modify the DOM with response data
  document.querySelector("#image1").src = response['
      message']
  )
 .catch(error =>
    // catch only handles net errors
    // (e.g when the server is down)
    console.err(error.message)
 );
```

Ejemplos de uso

Petición POST enviando datos en FormData

```
var formData = new FormData();
formData.append('username', username);
formData.append('password', password);

fetch('http://localhost:5000/login', {
  method: 'POST',
  body: formData
})
.then(response => {
  // Sucessful (200) and failure (4xx) http responses
  // are handled here
  if (response.status == 401) { // Error is handled in catch
    throw Error('Bad username or password');
  }
  return response.json();
})
.then(response => {
  token = response['access_token'];
  do_login();
})
.catch(error => document.querySelector('p#error-formpost').
    innerHTML = error.message);
```

Ejemplos de uso

Petición POST enviando datos en JSON

```
var data = {
  'username': username,
  'password': password
};

fetch('http://localhost:5000/login', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify(data)
})
.then(response => {
  if (response.status == 401) { // Error is handled in catch
    throw Error('Bad username or password');
  }
  return response.json();
})
.then(response => {
  token = response['access_token'];
  do_login();
})
.catch(error => document.querySelector('p#error-jsonpost').
    innerHTML = error.message);
```

Ejemplos de uso

Petición GET con autenticación vía token

```
var headers = new Headers();
//A token was sent in the response
// of the login request
headers.append('Authorization', 'Bearer ' + token);

fetch('http://localhost:5000/protected', {
  headers: headers
})
.then(response => response.json())
.then(response => {
  // The response only contains the username
  document.querySelector('p#response-ex2').innerHTML = '
      Logged in as ' + response['logged_in_as'];
})
.catch(error => console.err(error.message));
```

Ejemplos de uso

Petición GET con respuesta Blob

```
var headers = new Headers();
// Comment these headers to enable cache
headers.append('pragma', 'no-cache');
headers.append('cache-control', 'no-cache');

fetch('http://localhost:5000/image',{
  headers: headers,
})
.then(response =>
  // Returns a Promise to a Blob object
  response.blob()
)
.then(response => {
  // How to create an image from a Blob object
  var urlCreator = window.URL || window.webkitURL;
  var imageUrl = urlCreator.createObjectURL(response);
  document.querySelector("#image2").src = imageUrl;
})
.catch(error => console.err(error.message));
```