

UD 6

Comunicación asíncrona

DESARROLLO WEB EN ENTORNO CLIENTE

Técnico de Grado Superior Desarrollo de Aplicaciones Web

2024-25



J. Mario Rodríguez
jrodper183e@g.educaand.es

Contenidos

Utilización de mecanismos de comunicación asíncrona. AJAX:

- Mecanismos de comunicación asíncrona.
- Objetos, propiedades y métodos relacionados.
- Recuperación remota de información.
- Programación de aplicaciones con comunicación asíncrona.
- Modificación dinámica del documento utilizando comunicación asíncrona.
- Formatos para el envío y recepción de información.
- Librerías de actualización dinámica.
- JavaScript Object Notation. JSON.
- Uso de Fetch.
- Promesas.

Introducción

La comunicación asíncrona es una técnica fundamental en el desarrollo web moderno.

Permite actualizar partes de una página web sin necesidad de recargarla completamente, mejorando la experiencia del usuario.

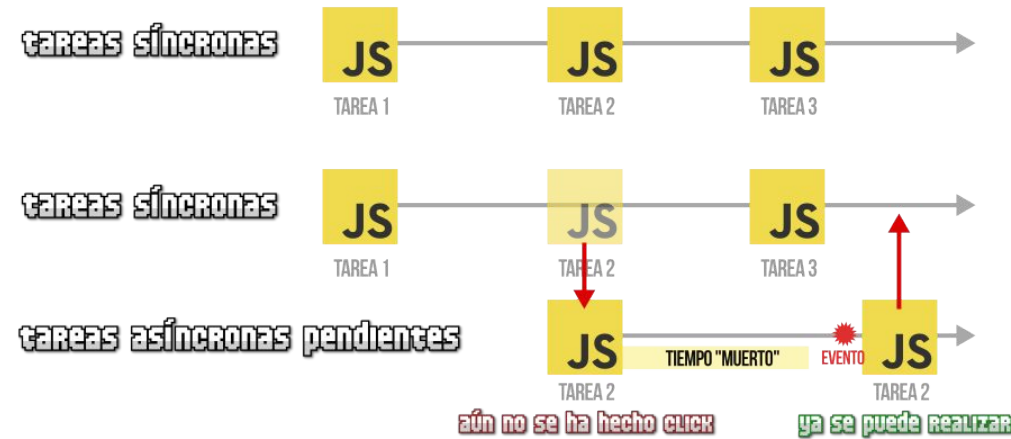
En esta unidad didáctica, aprenderemos cómo implementar mecanismos de comunicación asíncrona utilizando JavaScript, Fetch API, Promesas y JSON, entre otros conceptos clave.

Mecanismos de comunicación asíncrona

Conceptos que debemos tener claros:

- **Asincronía:**
Separación del flujo principal del programa
Se utiliza para realizar tareas que pueden tardar en completarse.
- **Petición HTTP:**
Comunicación entre cliente y servidor mediante [métodos HTTP](#)
- **Callback:**
Función que se ejecuta **después** de completar una tarea
Las [callbacks](#) se pasan como parámetro a otra función.

Mecanismos de comunicación asíncrona



- Pueden existir múltiples tareas asíncronas
- Las tareas pueden terminar realizandose correctamente
- Las tareas pueden terminar fallando y no realizarse
- Las tareas pueden quedarse pendientes de forma infinita y nunca realizarse o fallar.
- Las tareas pueden depender a su vez de otras tareas.
- Las tareas pueden tardar poco o tardar mucho.

Mecanismos de comunicación asíncrona

La comunicación asíncrona permite enviar y recibir datos del servidor sin bloquear el flujo de ejecución del programa.

¿Qué mecanismos veremos?

- **AJAX (Asynchronous JavaScript and XML)**
Técnica para actualizar partes de una página web dinámicamente.
- **Fetch API**
Una API moderna y más flexible que el antiguo objeto XMLHttpRequest.
- **Promesas**
Abstracción para manejar tareas asíncronas de manera ordenada.

Mecanismos de comunicación asíncrona

¿Qué formas veremos para manejar la asincronía?

Mecanismo	Descripción
Mediante callbacks	Probablemente, la forma más clásica de gestionar la asincronía.
Mediante promesas	Mecanismo moderno para gestionar la asincronía de forma no bloqueante.
Mediante <code>async/await</code>	Una forma simplificada de manejar promesas, pero bloqueante.

Mecanismos de comunicación asíncrona

Para abrir boca:

Vamos a realizar una petición simple con Fetch, así que creamos un html con un botón que haga una petición a una API pública y mostramos el resultado en un contenedor HTML.

```
<button id="fetchButton">Hacer llamada asíncrona...</button>  
<div id="output"></div>
```

```
document.getElementById('fetchButton').addEventListener('click', () => {  
  fetch(`...<ENDPOINT>...`)  
    .then(response => response.json())  
    .then(data => {  
      document.getElementById('output').innerHTML = `

### ${data.<campo1>}</h3><p>${data.<campo2>}</p>`; }) .catch(error => console.error('Error obteniendo los datos:', error)); });


```


Mecanismos de comunicación asíncrona

¿Qué es AJAX?

Es una técnica de desarrollo web, no un lenguaje.

AJAX significa:

- Asynchronous. El cliente hace peticiones asíncronas
- Javascript. Usando JS y el DOM para gestionar la petición
- And
- XML. la respuesta HTTP devuelve texto estructurado en HTML / XML o texto en crudo (text/plain) o organizado en JSON/YAML/CSV o similar

Algunos Inconvenientes

Usabilidad y Accesibilidad
Rendimiento
Seguridad
Complejidad (actual)



Objetos, propiedades y métodos relacionados

XMLHttpRequest

Aunque obsoleto en algunos contextos, es importante comprender su funcionamiento

Puede obtener cualquier tipo de dato, no solo XML

Proporciona una forma de obtener información de una URL sin tener que recargar la página.

XMLHttpRequest es ampliamente usado en la programación AJAX.



- `open(method, url)`: configura la petición.
- `send()`: envía la petición al servidor.
- `onreadystatechange`: evento que se activa al cambiar el estado de la petición.

Objetos, propiedades y métodos relacionados

Cómo implementar una petición con XMLHttpRequest

```
const xhr = new XMLHttpRequest();
xhr.open('GET', 'URL-endpoint', true);

xhr.onreadystatechange = function () {
  if (xhr.readyState === 4 && xhr.status === 200) {
    console.log(JSON.parse(xhr.responseText));
  }
};

xhr.send();
```

Value	State
0	UNSENT
1	OPENED
2	HEADERS_RECEIVED
3	LOADING
4	DONE

Códigos de estado de respuesta HTTP

Los códigos de estado de respuesta HTTP indican si se ha completado satisfactoriamente una solicitud HTTP específica. Las respuestas se agrupan en cinco clases:

- `http_request.responseText` – regresará la respuesta del servidor como una cadena de texto.
- `http_request.responseXML` – regresará la respuesta del servidor como un objeto `XMLDocument` que se puede recorrer usando las funciones de JavaScript DOM.

1. Respuestas informativas (100 – 199),
2. Respuestas satisfactorias (200 – 299),
3. Redirecciones (300 – 399),
4. Errores de los clientes (400 – 499),
5. y errores de los servidores (500 – 599).

Recuperación remota de información

La recuperación remota de datos implica:

- Establecer conexión con una API o servidor
- Procesar datos en formatos estándar (como JSON o XML)

	JSON	XML
Simplicidad	Más simple y ligero	Más complejo y verboso
Legibilidad	Más fácil de leer	Menos intuitivo
Uso	APIs modernas, intercambio de datos	Interoperabilidad entre sistemas
Velocidad	Rápido de procesar	Más lento debido a su estructura

Parsear es el proceso de analizar un formato de texto y convertirlo en una estructura de datos manejable dentro del programa.

Recuperación remota de información

De objeto JavaScript a JSON: `JSON.stringify()`

De JSON a objeto JavaScript: `JSON.parse()`

```
{  
  "nombre": "Juan",  
  "edad": 25,  
  "hobbies": ["fútbol", "lectura", "viajar"]  
}
```

XML requiere crear un DOM!

Usamos DOMParser para convertir cadenas de texto XML en un objeto DOM.

```
<persona>  
  <nombre>Juan</nombre>  
  <edad>25</edad>  
  <hobbies>  
    <hobby>fútbol</hobby>  
    <hobby>lectura</hobby>  
  </hobbies>  
</persona>
```

Aplicaciones con comunicación asíncrona

carga de datos dinámicos	obtener y mostrar datos en formatos como CSV, JSON o XML sin recargar la página.
integración con apis	consumir servicios externos para datos como clima, imágenes o pagos.
actualización en tiempo real	mostrar mensajes, notificaciones o datos en vivo sin interrupciones.
carga diferida de contenido	cargar imágenes o secciones solo cuando son visibles.
validación de formularios	verificar datos en tiempo real mediante el servidor.
actualización parcial del dom	modificar partes de la página sin recargarla completa.
cargas condicionales	descargar recursos solo cuando sean necesarios.
gestión de archivos	subir archivos con barras de progreso o descargar datos generados.
autocompletado y sugerencias	mostrar opciones predictivas al usuario.
páginas de configuración personalizada	cargar preferencias dinámicamente.
carga de mapas interactivos	integrar mapas con ubicaciones o búsquedas.
carros de compra	actualizar productos y totales sin recargar.
pagos y autenticación	validar datos como contraseñas o manejar pagos en tiempo real.
integración de redes sociales	cargar feeds o permitir publicaciones sin salir de la página.
aplicaciones multimedia	cargar progresivamente videos, audios o subtítulos.
gestión de juegos online	actualizar datos de partidas sin interrupciones.
aplicaciones basadas en geolocalización	mostrar servicios cercanos al usuario.
automatización de tareas en segundo plano	ejecutar procesos asíncronos como recordatorios.
aplicaciones basadas en inteligencia artificial	usar chatbots o sugerencias en tiempo real.
optimización de recursos del navegador	precargar datos antes de que sean necesarios.

Modificación dinámica del documento

es una de las capacidades más potentes de AJAX, ya que permite actualizar el contenido de una página web en tiempo real sin necesidad de recargarla

AJAX permite obtener datos desde un servidor que, una vez recibidos, pueden usarse para actualizar el DOM de la página web (texto, atributos, estructura...)



Usuarios

Cargar Usuarios

Reto:

- Leanne Graham (Sincere@april.biz)
- Ervin Howell (Shanna@melissa.tv)
- Clementine Bauch (Nathan@yesenia.net)
- Patricia Lebsack (Julianne.OConner@kory.org)
- Chelsey Dietrich (Lucio_Hettinger@annie.ca)
- Mrs. Dennis Schulist (Karley_Dach@jasper.info)
- Kurtis Weissnat (Telly.Hoeger@billy.biz)
- Nicholas Runolfsdottir V (Sherwood@rosamond.me)
- Glenna Reichert (Chaim_McDermott@dana.io)
- Clementina DuBuque (Rey.Padberg@karina.biz)

Formatos para enviar/recibir información

Principalmente



También

....



MIME Types

(Multipurpose Internet Mail Extensions)

indican el tipo de contenido enviado en una comunicación HTTP.

Cuando un servidor responde a una solicitud, incluye el tipo MIME en el *header* **Content-Type** para informar al cliente sobre cómo interpretar los datos.

```
Content-Type: application/json
```

- JSON: `application/json`
- XML: `application/xml` o `text/xml`
- HTML: `text/html`
- Texto plano: `text/plain`
- CSV: `text/csv`

Librerías de actualización dinámica



algunos beneficios en el manejo automático de errores, en el manejo del *timeout*, uso de *interceptors*, y facilidad de configuración en general

manejan las diferencias entre navegadores, asegurando que las aplicaciones funcionen en entornos diversos

```
axios.get('/endpoint', {  
  timeout: 5000, // Tiempo de espera  
  headers: { 'Authorization': 'Bearer token' }  
})  
.then(response => console.log(response.data))  
.catch(error => console.error(error.message));
```



añade un peso hoy día innecesario
otros frameworks solapan su cometido
es útil únicamente en proyectos *legacy*




está integrada de forma nativa en los navegadores modernos; es ligera y funciona bien con promesas, compatible con *async/await*.

JSON (JavaScript Object Notation)

<https://www.json.org/json-es.html>

<https://developer.mozilla.org/es/docs/Learn/JavaScript/Objects/JSON>

¿Qué es realmente JSON?

[JSON](#) es un formato de datos basado en texto que sigue la sintaxis de objeto de JavaScript, popularizado por [Douglas Crockford](#) . Aunque es muy parecido a la sintaxis de objeto literal de JavaScript, puede ser utilizado independientemente de JavaScript, y muchos entornos de programación poseen la capacidad de leer (convertir; *parsear*) y generar JSON.

Los JSON son cadenas - útiles cuando se quiere transmitir datos a través de una red. Debe ser convertido a un objeto nativo de JavaScript cuando se requiera acceder a sus datos. Ésto no es un problema, dado que JavaScript posee un objeto global [JSON](#) que tiene los métodos disponibles para convertir entre ellos.

Nota: Convertir una cadena a un objeto nativo se denomina *parsing*, mientras que convertir un objeto nativo a una cadena para que pueda ser transferido a través de la red se denomina *stringification*.

Un objeto JSON puede ser almacenado en su propio archivo, que es básicamente sólo un archivo de texto con una extensión `.json`, y una [MIME type](#) ^(inglés) de `application/json`.

```
{
  "squadName": "Super hero squad",
  "homeTown": "Metro City",
  "formed": 2016,
  "secretBase": "Super tower",
  "active": true,
  "members": [
    {
      "name": "Molecule Man",
      "age": 29,
      "secretIdentity": "Dan Jukes",
      "powers": ["Radiation resistance", "Turning tiny", "Radiation blast"]
    },
    {
      "name": "Madame Uppercut",
      "age": 39,
      "secretIdentity": "Jane Wilson",
      "powers": [
        "Million tonne punch",
        "Damage resistance",
        "Superhuman reflexes"
      ]
    },
    {
      "name": "Eternal Flame",
      "age": 1000000,
      "secretIdentity": "Unknown",
      "powers": [
        "Immortality",
        "Heat Immunity",
        "Inferno",
        "Teleportation",
        "Interdimensional travel"
      ]
    }
  ]
}
```

Fetch

es una interfaz moderna y nativa de los navegadores que permite realizar solicitudes HTTP para enviar o recibir datos

Reemplaza a XMLHttpRequest con una sintaxis más simple basada en promesas.

```
fetch('https://api.example.com/data')
  .then(response => {
    if (!response.ok) { // Manejo de errores HTTP
      throw new Error(`Error HTTP: ${response.status}`);
    }
    return response.json(); // Convertir respuesta a JSON
  })
  .then(data => {
    console.log('Datos recibidos:', data);
  })
  .catch(error => {
    console.error('Error en la solicitud:', error);
  });
```

```
fetch('https://api.example.com/submit', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json' // MIME Type para JSON
  },
  body: JSON.stringify({ name: 'Juan', age: 25 })
})
  .then(response => response.json())
  .then(data => console.log('Respuesta del servidor:', data))
  .catch(error => console.error('Error:', error));
```

Promesas

mecanismo para manejar tareas asíncronas que representa un valor que puede estar disponible **ahora**, en el **futuro**, o **nunca**.

Estados de una promesa:

pending: La operación aún no ha finalizado.

fulfilled: La operación fue exitosa y se resolvió con un valor.

rejected: La operación falló y se resolvió con un error.

Promesas

```
const promesa = new Promise((resolve, reject) => {
  setTimeout(() => {
    const success = true; // Simulación de éxito o error
    if (success) {
      resolve('¡Operación exitosa!');
    } else {
      reject('Algo salió mal.');
```



```
    }
  }, 2000);
});

// Uso de La promesa
promesa
  .then(result => console.log(result)) // Se ejecuta si se resuelve
  .catch(error => console.error(error)) // Se ejecuta si hay error
  .finally(() => console.log('Operación finalizada')); // Siempre se ejecuta
```

async-await

es una sintaxis introducida en ES2017 para trabajar con promesas de una forma más clara y similar a código síncrono.

Características:

Las funciones marcadas con `async` devuelven una promesa.

`await` pausa la ejecución hasta que la promesa se resuelve o rechaza.

Simplifica la lectura de código, especialmente si hay múltiples llamadas asíncronas.



Evita el "callback hell"

async-await

```
async function obtenerDatos() {  
  try {  
    const response = await fetch('https://api.example.com/data');  
    if (!response.ok) {  
      throw new Error(`Error HTTP: ${response.status}`);  
    }  
    const data = await response.json(); // Espera a que se resuelva  
    console.log('Datos recibidos:', data);  
  } catch (error) {  
    console.error('Error en la solicitud:', error);  
  }  
}  
  
obtenerDatos();
```

