cedia | TMO Training Management Office

info@cedia.org.ec
cedia.edu.ec

CURSO DE JAVASCRIPT

Corporación Ecuatoriana para el Desarrollo de la Investigación y la Academia

Ing. María Fernanda Granda J., PhD

2-11-2025

“Conectando ideas, transformando sociedades”

Módulo 8: Programación basada en promesas.

- Definición de una promesa.
- Tratamiento del cumplimiento de una promesa.
- Áreas de aplicación

Definición de una Promesa

- Una promesa es un objeto que representa la finalización (o falla) eventual de una operación **asincrónica**. Esto evita que la interfaz de usuario se congele y permite una mejor experiencia del usuario.
- Permite trabajar con código que no se ejecuta de inmediato (como peticiones a un servidor), sin bloquear el resto del programa, como las peticiones HTTP, de forma más eficiente y organizada.

¿Qué son operaciones síncronas y asíncronas?

Operaciones Síncronas:

- En un modelo síncrono, si tu código necesita obtener datos de un servidor, tendría que **esperar a que la respuesta llegue antes de continuar** con la ejecución. Esto puede causar que la página se congele durante la espera.

Operaciones Asíncronas:

- Con las operaciones asíncronas, tu código puede iniciar la solicitud de red (la llamada HTTP) y luego continuar ejecutando otras tareas. Cuando la respuesta llega, un "callback" (una función que se ejecuta después de que la operación asíncrona se completa) maneja la respuesta, **sin bloquear la ejecución principal**.

Ventajas de las operaciones asíncronas en HTTP con JS

Mejora la experiencia del usuario: Al evitar bloqueos, la interfaz de usuario permanece receptiva, permitiendo al usuario interactuar con la página mientras se realizan las solicitudes de red.

Mayor eficiencia: Permite que el navegador realice otras tareas mientras espera la respuesta del servidor, lo que puede mejorar el rendimiento general de la aplicación.

Permite la multitarea: Puedes realizar múltiples solicitudes de red simultáneamente sin que se bloqueen entre sí.

Mecanismos comunes para realizar operaciones asíncronas en JS

Callbacks: Son funciones que se pasan como argumentos a otras funciones y se ejecutan una vez que la operación asíncrona ha terminado.

Promesas: Ofrecen una forma más estructurada y fácil de manejar operaciones asíncronas, permitiendo encadenar varias operaciones y manejar errores de manera más eficiente.

Async/Await:

Una sintaxis más moderna que facilita la escritura y lectura de código asíncrono, construida sobre las promesas.

¿Cómo funciona una promesa en JS?

- **Creación**: Se crea una promesa usando el constructor new Promise(). Este constructor toma una función con dos argumentos: resolve y reject.

```
let promesa = new Promise(function(resolve, reject) {
    //  Lógica asíncrona aquí
    // Si la operación es exitosa:
    resolve(valor);
    // Si la operación falla:
    // reject(error);
});
```

Ejemplo de promesa

- new Promise crea una promesa que se resuelve (o rechaza) después de 2 segundos.
- Si exito es true, se llama a resolve y se ejecuta la función de éxito en el then.
- Si exito es false, se llama a reject y se ejecuta la función de error en el then.

```
const miPromesa = new Promise((resolve, reject) => {
  // Simulación de una operación asíncrona que tarda 2 segundos
  setTimeout(() => {
    const exito = true; // Cambiar a false para simular un error
    if (exito) {
      resolve("La operación fue exitosa");
    } else {
      reject(new Error("La operación falló"));
    }
  }, 2000);
});

miPromesa.then(
  (resultado) => {
    console.log("Éxito:", resultado); // Se ejecutará si resolve es llamado
  },
  (error) => {
    console.error("Error:", error); // Se ejecutará si reject es llamado
  }
);
```

Ejemplo de promesa

- La función **fetch** en JavaScript se utiliza para realizar solicitudes de red, como obtener datos de una API o enviar datos a un servidor es soportado por la mayoría de los navegadores modernos.

- `fetch()` devuelve una promesa que representa la petición HTTP.
- `.then()` se usa para procesar la respuesta. Si la respuesta no es exitosa (por ejemplo, un código de estado 404), se lanza un error que es capturado por el `catch()`.
- `.then()` se usa para convertir la respuesta a JSON.
- `.then()` finalmente maneja los datos JSON.
- `.catch()` captura cualquier error que ocurra en la cadena de promesas.

```
fetch('https://jsonplaceholder.typicode.com/users')
    .then(response => {
        if (!response.ok) {
            throw new Error('La red no responde');
        }
        return response.json(); //  Convertir a JSON
    })
    .then(data => {
        console.log(data); // Manejar los datos
    })
    .catch(error => {
        console.error('Hubo un problema con la operación fetch:', error); // Manejar errores
    });
```

¿Qué es JSON?

- JSON, o Notación de Objetos de JavaScript, es un formato ligero para el intercambio de datos.
- Utiliza pares clave-valor y estructuras de datos como arreglos para representar información.
- Se utiliza comúnmente para transmitir datos en aplicaciones web, actuando como una alternativa más simple y liviana al XML.

```
{
  "nombre": "Juan Pérez",
  "edad": 30,
  "ciudad": "Madrid",
  "intereses": ["música", "libros", "deportes"]
}
```

Estados de una Promesa

Una promesa puede estar en tres estados:

- **pending** (pendiente): aún no se ha completado ni fallado.
- **fulfilled** (cumplida): la operación se completó con éxito.
- **rejected** (rechazada): la operación falló.

Manejo de Resultados

Se utilizan los métodos then() y catch() para manejar los resultados de la promesa.

.then(onFulfilled, onRejected): Se ejecuta cuando la promesa se cumple. **onFulfilled** es una función que se ejecuta con el valor de la promesa cumplida. **onRejected** es una función que se ejecuta con el error en caso de que la promesa sea rechazada.

.catch(onRejected): Se ejecuta cuando la promesa es rechazada. **onRejected** es una función que maneja el error.

Encadenamiento de promesas

- Se pueden encadenar múltiples then() para ejecutar operaciones secuenciales después de que una promesa se cumpla.
- Cada then() devuelve una nueva promesa, permitiendo una estructura de código más clara para operaciones asíncronas compleja.

Ejemplo de Encadenamiento de promesas

- En este caso, si la primera promesa es rechazada, el .catch() captura el error, y el resto de la cadena (los .then() posteriores) no se ejecutan. Si no hubiera un .catch(), el error se propagaría y podría interrumpir la ejecución del código.

```
new Promise(function(resolve, reject) {
  setTimeout(() => resolve(1), 1000); // Resuelve con 1 después de 1 segundo
})
.then(function(result) {
  console.log(result);  // 1
  return result * 2;    // Devuelve el valor multiplicado por 2
})
.then(function(newResult) {
  console.log(newResult); // 2
  return newResult + 3;   // Devuelve el valor sumado a 3
})
.then(function(finalResult) {
  console.log(finalResult); // 5
})
.catch(function(error) {
  console.error("Hubo un error:", error); // Captura el error
})
.then(function(result) {
    console.log("Este then no se ejecuta si hay error");
});
```

Ventajas de usar promesas

- **Legibilidad**: Las promesas permiten escribir código asíncrono más legible y estructurado".
- **Manejo de errores**: Proporcionan un mecanismo más claro y consistente para manejar errores en operaciones asíncronas.
- **Composición**: Permiten encadenar operaciones asíncronas de manera más sencilla, mejorando la mantenibilidad del código.
- **Soporte amplio**: Son una característica estándar en JavaScript moderno y son soportadas por la mayoría de los navegadores y entornos JavaScript.

Áreas de aplicación

Llamadas a APIs:

- Las promesas son ideales para realizar peticiones a servidores web y manejar las respuestas, ya sea que la petición sea exitosa o falle.

Lectura y escritura de archivos:

- Al trabajar con sistemas de archivos, las promesas permiten manejar la lectura y escritura de datos de manera asíncrona.

Cualquier operación que pueda tardar:

- En general, cualquier tarea que pueda tomar tiempo y que necesite ejecutarse de forma asíncrona puede ser manejada con promesas.

Áreas de aplicación

Manejo de errores:

Permiten capturar y manejar errores que puedan ocurrir durante la ejecución de una operación asíncrona, evitando que el programa falle inesperadamente.

Encadenamiento de operaciones:

Permiten encadenar varias operaciones asíncronas, donde el resultado de una operación se utiliza como entrada para la siguiente.

Async/Await:

Las promesas son la base para usar las palabras clave async y await, que simplifican aún más la escritura de código asíncrono.

cedia | TMO Training Management Office

GRACIAS

cedia.edu.ec
info@cedia.org.ec
(+593) 7-407-9300