



Bootcamp de Desarrollo Web

Clase 11

Uso de bibliotecas de JavaScript para trabajar con API

1



Javascript runtime

El **JavaScript runtime** (tiempo de ejecución de JavaScript) es el entorno en el que se ejecuta el código JavaScript.

Este entorno proporciona el contexto en el que se realizan las operaciones de tiempo de ejecución, como la ejecución de código, la administración de memoria, el manejo de eventos, el acceso a objetos del navegador (en el caso de JavaScript en el navegador web), etc.

Bootcamp Desarrollo Web

2



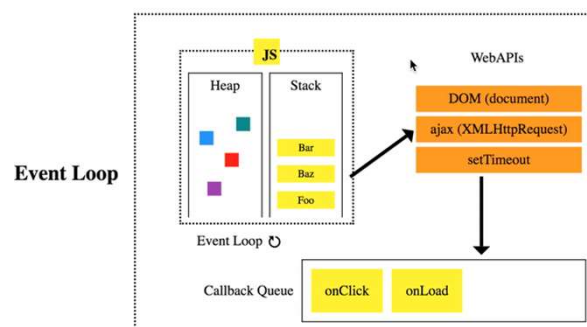
¿Javascript es síncrono o asíncrono?

JS es un lenguaje asíncrono y no bloqueante, esto porque nuestro espacio de ejecución solo puede responder a 1 tarea a la vez pero pudiendo hacerlas sin importar si la respuesta es dato o error con tal de no obstruir el programa

3



Javascript runtime



4



Partes del JS Runtime

Call Stack (Pila de Llamadas):

- La pila de llamadas es una estructura de datos que se utiliza para gestionar el flujo de ejecución en JavaScript.
- Cada vez que una función se llama, un nuevo marco de pila se agrega a la parte superior de la pila.
- Cuando una función termina de ejecutarse, su marco de pila se elimina de la parte superior de la pila.



Partes del JS Runtime

Event Loop (Bucle de Eventos):

- El bucle de eventos es el mecanismo central que permite la ejecución asíncrona en JavaScript.
- Es responsable de manejar la ejecución de eventos y tareas asíncronas.
- Continuamente revisa la pila de llamadas y la cola de devolución de llamada (callback queue) para determinar si hay funciones que se pueden ejecutar.
- Si la pila de llamadas está vacía, el bucle de eventos toma la primera función de la cola de devolución de llamada y la agrega a la pila de llamadas para su ejecución.



Partes del JS Runtime

Heap (Montículo):

- El montículo es donde se almacenan las variables y objetos asignados dinámicamente durante la ejecución del programa.
- Es una región de memoria no estructurada y dinámica que se utiliza para almacenar datos.



Partes del JS Runtime

Callback Queue (Cola de Devolución de Llamada):

- La cola de devolución de llamada es donde se colocan las funciones de devolución de llamada después de que se completan las operaciones asíncronas.
- Cuando se completa una tarea asíncrona, su función de devolución de llamada (callback) se coloca en la cola de devolución de llamada para ser procesada por el bucle de eventos.



Partes del JS Runtime

APIs (Interfaz de Programación de Aplicaciones):

- Las APIs proporcionan funcionalidades adicionales que no están disponibles directamente en JavaScript, como acceso al DOM en el navegador o a funciones del sistema en Node.js.
- Las operaciones asíncronas, como solicitudes HTTP, temporizadores, eventos del DOM, etc., son manejadas a través de estas APIs.
- Cuando se completa una operación asíncrona, su función de devolución de llamada (callback) se coloca en la cola de devolución de llamada para ser procesada por el bucle de eventos.

Bootcamp Desarrollo Web

9






Partes del JS Runtime

APIs (Interfaz de Programación de Aplicaciones):

- Las APIs proporcionan funcionalidades adicionales que no están disponibles directamente en JavaScript, como acceso al DOM en el navegador o a funciones del sistema en Node.js.
- Las operaciones asíncronas, como solicitudes HTTP, temporizadores, eventos del DOM, etc., son manejadas a través de estas APIs.
- Cuando se completa una operación asíncrona, su función de devolución de llamada (callback) se coloca en la cola de devolución de llamada para ser procesada por el bucle de eventos.

Bootcamp Desarrollo Web



10



Bootcamp de Desarrollo Web

**Javascript
Promises**

11



Bibliotecas

Una biblioteca, en el contexto de la programación de software, es un conjunto de funciones, clases y rutinas que están diseñadas para ser reutilizadas por otros programas.

Estas funciones y rutinas están organizadas y empaquetadas de manera que puedan ser fácilmente incorporadas en proyectos de software más grandes.

Bootcamp Desarrollo Web

12



Ventajas de usar bibliotecas

Reutilización de código: Permite a los desarrolladores reutilizar el código existente en lugar de tener que volver a escribirlo desde cero para cada proyecto.

Eficiencia: Al utilizar una biblioteca probada y bien mantenida, los desarrolladores pueden ahorrar tiempo y esfuerzo en el desarrollo de software.

Consistencia: Las bibliotecas bien diseñadas proporcionan una interfaz coherente y predecible, lo que facilita su uso y reduce la posibilidad de errores.

Bootcamp Desarrollo Web

13



Ventajas de usar bibliotecas

Estandarización: Algunas bibliotecas se han convertido en estándares de facto en la industria y son ampliamente utilizadas y reconocidas por los desarrolladores.

Comunidad y soporte: Las bibliotecas populares suelen tener una gran base de usuarios y una comunidad activa que proporciona soporte, documentación y ejemplos de uso.

Bootcamp Desarrollo Web

14



Bibliotecas para consumir API

Existen muchas bibliotecas JavaScript populares que se utilizan para trabajar con APIs de manera eficiente.

Estas bibliotecas proporcionan funciones y utilidades útiles para realizar solicitudes HTTP, manejar respuestas de API, y simplificar la interacción con servicios web.

Algunas de las bibliotecas más comunes son:



Bibliotecas para consumir API

Fetch API:

- La Fetch API es una API nativa de JavaScript para realizar solicitudes HTTP.
- Proporciona una interfaz moderna y basada en promesas para realizar solicitudes AJAX.
- Es compatible con la mayoría de los navegadores modernos, pero es posible que necesites usar un polyfill para navegadores más antiguos.



Bibliotecas para consumir API

Axios:

- Axios es una biblioteca popular para hacer solicitudes HTTP desde el navegador y Node.js.
- Proporciona una interfaz simple y fácil de usar para realizar solicitudes GET, POST, PUT, DELETE, etc.
- Soporta promesas para trabajar de manera asíncrona.
- Es ampliamente utilizado debido a su simplicidad y versatilidad.



Bibliotecas para consumir API

jQuery (para aplicaciones más antiguas):

- jQuery es una biblioteca JavaScript muy popular que simplifica la manipulación del DOM y la realización de solicitudes AJAX.
- Aunque su uso ha disminuido en aplicaciones modernas debido a la adopción de estándares más modernos, sigue siendo una opción viable para proyectos más antiguos o aquellos que ya están utilizando jQuery en otras partes del código.



Bootcamp de Desarrollo Web

Autenticación

19



¿Qué es la autenticación de API?

La autenticación de API es el proceso de verificar la identidad de un cliente (ya sea una aplicación o un usuario) que intenta acceder a una API.

Permite controlar y limitar el acceso a los recursos protegidos de una API, garantizando que solo los usuarios autorizados puedan realizar solicitudes y obtener datos.

Bootcamp Desarrollo Web

20



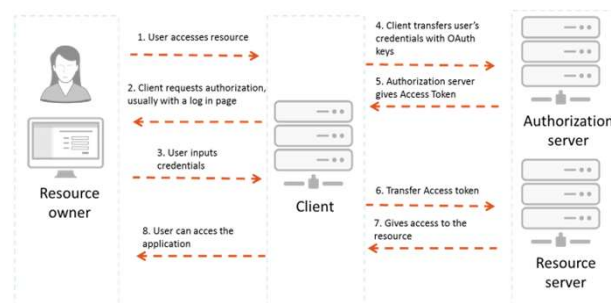
Cómo utilizar claves de API para autenticarse en una API

- Las claves de API son cadenas de caracteres generadas por el proveedor de la API y asignadas a cada cliente que desea acceder a la API.
- Cuando un cliente realiza una solicitud a la API, incluye su clave de API en la solicitud para identificarse.
- La API verifica la clave de API incluida en la solicitud para garantizar que sea válida y coincide con una clave autorizada en su sistema.
- Si la clave de API es válida, la API permite al cliente acceder a los recursos protegidos y procesa la solicitud; de lo contrario, devuelve un error de autenticación.



Que es OAuth

OAuth es un protocolo de autorización que permite que una aplicación obtenga acceso a los recursos en nombre de un usuario sin compartir las credenciales de este usuario.





Cómo utilizar OAuth para autenticarse en una API

- En OAuth, un usuario autentica una aplicación en un proveedor de servicios y otorga permiso a la aplicación para acceder a sus datos en el proveedor de servicios.
- La autenticación se realiza mediante tokens de acceso que se intercambian entre la aplicación y el proveedor de servicios.
- El flujo típico de OAuth involucra la redirección del usuario al proveedor de servicios, donde autentica la aplicación y otorga permisos. Luego, el proveedor de servicios redirige al usuario de vuelta a la aplicación con un código de autorización. La aplicación intercambia este código por un token de acceso que puede usar para realizar solicitudes en nombre del usuario.

Bootcamp Desarrollo Web

23



Bootcamp de Desarrollo Web

**Javascript
Promises**

24



Tipo de operaciones

En JavaScript, las operaciones pueden ser síncronas o asíncronas, dependiendo de cómo se ejecuten en relación con el flujo principal de ejecución del programa.

Síncrono (Synchronous):

- Las operaciones síncronas se ejecutan una tras otra en secuencia.
- El programa espera a que una operación síncrona se complete antes de pasar a la siguiente.
- Esto significa que el flujo de ejecución está bloqueado hasta que la operación actual se complete.



Ejemplo de programación Síncrona

```
console.log('Inicio');  
console.log('Hola');  
console.log('Mundo');  
console.log('Fin');
```

En este ejemplo, las instrucciones se ejecutan secuencialmente, y **Inicio**, **Hola**, **Mundo** y **Fin** se imprimirán en ese orden.



Tipo de operaciones

Asíncrono (Asynchronous):

- Las operaciones asíncronas no bloquean el flujo principal de ejecución.
- En lugar de esperar a que una operación se complete, el programa continúa ejecutando otras operaciones mientras espera el resultado de la operación asíncrona.
- Cuando la operación asíncrona se completa, se ejecuta una función de retorno de llamada (**callback**) o se resuelve una promesa, dependiendo del mecanismo utilizado para manejar la asincronía.



Ejemplo de programación Asíncrono - callback

```
console.log('Inicio');
setTimeout(() => {
  console.log('¡Hola después de 2 segundos!');
}, 2000);
console.log('Fin');
```

En este ejemplo, **Inicio** y **Fin** se imprimirán inmediatamente, mientras que **¡Hola después de 2 segundos!** se imprimirá después de una espera de 2 segundos.



Promesas

En JavaScript, una **promesa** es un **objeto** que representa el resultado eventual de una operación asíncrona.

Las promesas se utilizan para manejar tareas asíncronas de manera más legible y más fácilmente mantenible en comparación con el uso de **callbacks** anidados.



Estados de una promesa

Una promesa puede estar en uno de tres estados:

Pendiente (pending): Cuando la operación asíncrona aún no se ha completado.

Cumplida (fulfilled): Cuando la operación asíncrona se ha completado con éxito.

Rechazada (rejected): Cuando la operación asíncrona ha fallado o ha sido rechazada.



Sintaxis de una promesa

La sintaxis básica para crear una promesa es la siguiente:

```
const miPromesa = new Promise((resolve, reject) => {  
  // Realiza alguna operación asíncrona  
  // Si la operación tiene éxito, llama a resolve con el resultado  
  // Si la operación falla, llama a reject con el motivo del fallo  
});
```



Ejemplo de una promesa

```
const miPromesa = new Promise((resolve, reject) => {  
  setTimeout(() => {  
    const exito = true;  
    if (exito) {  
      resolve('¡La operación fue exitosa!');  
    } else {  
      reject('¡La operación falló!');  
    }  
  }, 2000); // Simula una operación asíncrona que tarda 2 segundos  
});
```




Consumir promesa

Para consumir el resultado de una promesa, puedes encadenar llamadas a los métodos **.then()** y **.catch()**:

En este ejemplo, si la promesa se resuelve exitosamente, el método **.then()** se ejecutará con el resultado. Si la promesa es rechazada, el método **.catch()** capturará el motivo del rechazo.

```
miPromesa
  .then(resultado => {
    console.log('Resultado:', resultado);
  })
  .catch(error => {
    console.error('Error:', error);
  });
```

Bootcamp Desarrollo Web

33



Método then() y catch()

El método **.then()** y **.catch()** son métodos utilizados para manejar el resultado de una promesa en JavaScript.

Estos métodos son parte de la API de las promesas y proporcionan una forma elegante de trabajar con el resultado eventual de una operación asíncrona.

Bootcamp Desarrollo Web

34



Método then() y catch()

then():

- El método .then() se utiliza para manejar el caso en el que una promesa es resuelta exitosamente.
- Toma uno o dos argumentos: una función de retorno de llamada que se ejecutará cuando la promesa se resuelva y un segundo argumento opcional para manejar el rechazo de la promesa.
- La función de retorno de llamada que se pasa al método .then() recibe el valor resuelto de la promesa como su argumento.



Ejemplo then()

```
const miPromesa = new Promise((resolve, reject) => {  
  // Simulando una operación asíncronica exitosa  
  setTimeout(() => {  
    resolve('¡Operación exitosa!');  
  }, 2000);  
});  
  
miPromesa.then(resultado => {  
  console.log('Resultado:', resultado);  
});
```

En este ejemplo, la función de retorno de llamada dentro del método .then() se ejecutará cuando la promesa sea resuelta con éxito, recibiendo el valor **Operación exitosa** como su argumento.



Método then() y catch()

catch():

- El método .catch() se utiliza para manejar el caso en el que una promesa es rechazada.
- Toma una función de retorno de llamada como argumento que se ejecutará cuando la promesa sea rechazada.
- La función de retorno de llamada que se pasa al método .catch() recibe el motivo del rechazo como su argumento.



Ejemplo catch()

```
const miPromesa = new Promise((resolve, reject) => {  
  // Simulando una operación asincrónica que falla  
  setTimeout(() => {  
    reject('¡Operación fallida!');  
  }, 2000);  
});  
  
miPromesa  
  .then(resultado => {  
    console.log('Resultado:', resultado);  
  })  
  .catch(error => {  
    console.error('Error:', error);  
  });
```

En este ejemplo, la función de retorno de llamada dentro del método .catch() se ejecutará si la promesa es rechazada, recibiendo el motivo del rechazo (**¡Operación fallida!**) como su argumento.



Ejemplo Asíncrono - promesas

```
console.log('Inicio');
const miPromesa = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve('¡Hola después de 2 segundos!');
  }, 2000);
});
miPromesa.then(resultado => {
  console.log(resultado);
});
console.log('Fin');
```

En este caso, **Inicio** y **Fin** se imprimirán inmediatamente, y **¡Hola después de 2 segundos!** se imprimirá después de una espera de 2 segundos, pero gestionado mediante promesas.

Bootcamp Desarrollo Web

39



Bootcamp de Desarrollo Web

Clase 11

Async Await

40



Async/Await

En JavaScript, las palabras clave `async` y `await` se utilizan en conjunto para trabajar con funciones asíncronas de manera más legible y manejable.

async: Se usa para declarar que una función es asíncrona. Esto significa que la función ejecutará operaciones de manera asíncrona y devolverá una promesa.

```
async function miFuncionAsincrona() {  
  // Código asíncrono aquí  
}
```

Bootcamp Desarrollo Web

41



Async/Await

await: Se utiliza dentro de una función `async` para esperar la resolución de una promesa. Permite que el código espere hasta que la promesa se resuelva y luego continúe su ejecución.

```
async function miFuncionAsincrona() {  
  let resultado = await promesa;  
  // El código espera a que la promesa se resuelva antes de continuar  
}
```

Bootcamp Desarrollo Web

42



Async/Await

Usando async y await juntos, podemos escribir código asíncrono de manera más limpia y fácil de entender. Por ejemplo:

```
async function obtenerDatosUsuario(idUsuario) {  
  try {  
    let usuario = await buscarUsuarioEnBaseDeDatos(idUsuario);  
    let historial = await obtenerHistorialDeUsuario(idUsuario);  
    return { usuario, historial };  
  } catch (error) {  
    console.error("Error al obtener datos del usuario:", error);  
    throw error;  
  }  
}  
  
obtenerDatosUsuario(123)  
  .then(datos => console.log(datos))  
  .catch(error => console.error(error));
```

Bootcamp Desarrollo Web

43



Async/Await

- En el ejemplo anterior, la función obtenerDatosUsuario es una función asíncrona que espera la resolución de dos promesas (buscarUsuarioEnBaseDeDatos y obtenerHistorialDeUsuario).
- Usando await, esperamos a que cada una de estas promesas se resuelva antes de continuar con la ejecución de la función. Esto hace que el código sea más legible y más similar a la programación síncrona.

Bootcamp Desarrollo Web

44



Bootcamp de Desarrollo Web

Clase 11

**Uso de bibliotecas de JavaScript
para trabajar con API**