UD3 – Programación Asíncrona

DAW2 - DWEC

JavaScript mono hilo – pero…

Sólo se ejecuta una tarea a la vez en el hilo principal.

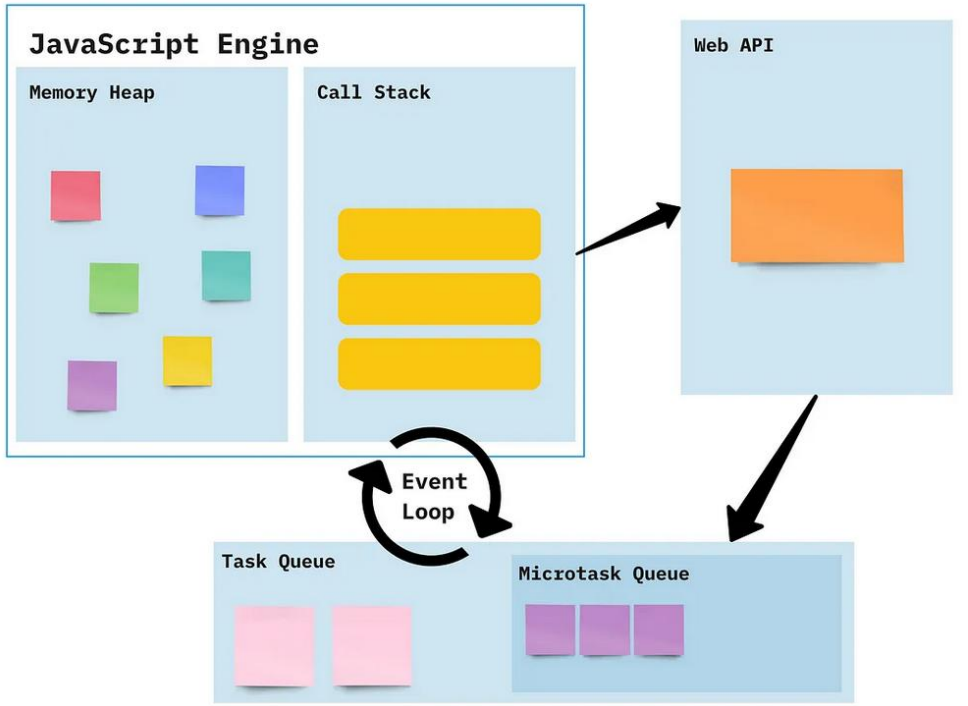
Modelo de asincronía basado en el “**evento loop**”.

1. Microtareas: queueMicrotask()
2. Tareas: setTimeout(), setInterval(), eventos

La asincronía permite gestionar tareas que requieren tiempo.

Herramientas

1. Callbacks
2. Promesas
3. Async/Await



Callbacks

Funciones que se pasan como argumento a otras funciones y se ejecutan después de que estas completen su tarea.

```
console.log("Inicio");
    setTimeout(() => {
        console.log("Esto se ejecuta después de 2 segundos");
    }, 2000);
console.log("Fin");
```

PROBLEMA: **Callback Hell** (código anidado y difícil de mantener).

Callback Hell

Múltiples callbacks anidados, creando un código poco legible.

```
realizarTarea1((resultado1) => {
    realizarTarea2(resultado1, (resultado2) => {
        realizarTarea3(resultado2, (resultado3) => {
            console.log("Resultado final", resultado3);
        });
    });
});
```

SOLUCIÓN: Uso de **Promesas** y posteriormente **Async/Await**.

Promesas - Promise

Objeto que representa el eventual éxito o fracaso de una operación asíncrona.

Estados (se gestionan a través de métodos)

- **pending**: En espera.
- **fulfilled**: Resuelta con éxito.
- **rejected**: Fallida.

```
const promesa = new Promise((resolve, reject) => {
    const exito = true;
    if (exito) {
        resolve("Operación exitosa");
    } else {
        reject("Hubo un error");
    }
});
```

Uso de promesas

Uso:

- **.then** => resolve
- **.catch** => rejected
- **.finally** => siempre

```
promesa.then((resultado) => console.log(resultado))
    .catch((error) => console.error(error))
    .finally(() => console.log("Siempre se ejecuta"));
```

Las promesas facilitan la legibilidad y la gestión de errores.

Promesas - Métodos adicionales

Promise.resolve(valor): Crea una promesa resuelta.

Promise.reject(error): Crea una promesa rechazada.

Promise.all([promesa1, promesa2]): Ejecuta múltiples promesas en paralelo. Espera hasta que todas tienen éxito o hasta la primera que falla.

```
Promise.all([promesa1, promesa2])
    .then((resultados) => console.log(resultados))
    .catch((error) => console.error(error));
```

Async y Await

Permite escribir código asíncrono como si fuera sincrónico.

Una función **async** devuelve siempre una promesa.

El uso de **await** pausa la ejecución hasta que la promesa se resuelva o rechace.

```
async function obtenerUsuario() {
    try {
        const usuario = await obtenerDatos();
        console.log("Usuario obtenido:", usuario);
    } catch (error) {
        console.error("Error:", error);
    } finally {
        console.log("Finalizando operación");}
}
obtenerUsuario();
```

Preguntas