Async/Await - Javascript en español

En se introducen las palabras clave `async`/`await`, que no son más que una forma de **azúcar sintáctico** para gestionar las promesas de una forma más sencilla. Con `async`/`await` seguimos utilizando promesas, pero abandonamos el modelo de encadenamiento de `.then()` para utilizar uno en el que trabajamos de forma más tradicional.

La palabra clave async

En primer lugar, tenemos la palabra clave `async`. Esta palabra clave se colocará previamente a `function`, para definirla así como una **función asíncrona**, el resto de la función no cambia:

```
async function funcion_asincrona() {
  return 42;
}
```

En el caso de que utilicemos **arrow function**, se definiría como vemos a continuación, colocando el `async` justo antes de los parámetros de la arrow function:

```
const funcion_asincrona = async () => 42;
```

Al ejecutar la función veremos que ya nos devuelve una que ha sido cumplida, con el valor devuelto en la función (*en este caso, 42*). De hecho, podríamos utilizar un `.then()` para manejar la promesa:

```
funcion_asincrona().then(value => {
  console.log("El resultado devuelto es: ", value);
});
```

Sin embargo, veremos que lo que se suele hacer junto a `async` es utilizar la palabra clave `await`, que es donde reside lo interesante de utilizar este enfoque.

La palabra clave await

Cualquier función definida con `async`, o lo que es lo mismo, cualquier puede utilizarse junto a la palabra clave `await` para manejarla. Lo que hace `await` es esperar a que se resuelva la promesa, mientras permite continuar ejecutando otras tareas que puedan realizarse:

```
const funcion_asincrona = async () => 42;

const value = funcion_asincrona();             // Promise { <fulfilled>: 42 }
const asyncValue = await funcion_asincrona();  // 42
```

Observa que en el caso de `value`, que se ejecuta sin `await`, lo que obtenemos es el valor devuelto por la función, pero «envuelto» en una promesa que deberá utilizarse con `.then()` para manejarse. Sin embargo, en `asyncValue` estamos obteniendo un tipo de dato , guardando el valor directamente ya procesado, ya que `await` espera a que se resuelva la promesa de forma asíncrona y guarda el valor.

Esto hace que la forma de trabajar con `async`/`await`, aunque se sigue

trabajando exactamente igual con promesas, sea mucho más fácil y trivial para usuarios que no estén acostumbrados a las promesas y a la asincronía en general, ya que el código «parece» síncrono.

Asincronía con async/await

Volvamos al ejemplo que hemos visto en los anteriores capítulos. Recordemos que la función `doTask()` realiza 10 lanzamientos de un dado y nos devuelve los resultados obtenidos o detiene la tarea si se obtiene un 6.

La implementación de la función sufre algunos cambios, simplificándose considerablemente. En primer lugar, añadimos la palabra clave `async` antes de los parámetros de la **arrow function**. En segundo lugar, desaparece cualquier mención a promesas, se devuelven directamente los objetos, ya que al ser una función `async` se devolverá todo envuelto en una :

```
const doTask = async (iterations) => {
  const numbers = [];
  for (let i = 0; i < iterations; i++) {
    const number = 1 + Math.floor(Math.random() * 6);
    numbers.push(number);
    if (number === 6) {
      return {
        error: true,
        message: "Se ha sacado un 6"
      };
    }
  }
  return {
    error: false,
    value: numbers
  };
}
```

Pero donde se introducen cambios considerables es a la hora de consumir las promesas con `async`/`await`. No tendríamos que utilizar `.then()`, sino que podemos simplemente utilizar `await` para esperar la resolución de la promesa, obteniendo el valor directamente:

```
const resultado = await doTask(10);   // Devuelve un objeto, no una promesa
```

> Recuerda que en el caso de querer controlar errores o promesas rechazadas, siempre podrás utilizar bloques `try`/`catch`.

Top-level await

En principio, el comportamiento de `await` solo permite que se utilice en el interior de funciones declaradas como `async`. Por lo que, si el ejemplo anterior lo ejecutamos en una consola de Javascript, funcionará correctamente (*estamos escribiendo comandos de forma asíncrona*), pero si lo escribimos en un fichero, probablemente nos aparecerá el siguiente error:

> Uncaught SyntaxError: `await` is only valid in `async` function

Esto ocurre porque, como bien dice el mensaje de error, estamos ejecutando `await` en el contexto global de la aplicación, y debemos ejecutarlo en un contexto de función asíncrona. Para corregirlo, podemos añadir un `<button>` en el HTML y modificar la linea anterior del `await`:

```
document.querySelector("button").addEventListener("click", async () => {
  const resultado = await doTask(10);
  console.log(resultado);
});
```

Una nueva propuesta denominada top-level await permite utilizar `await` fuera de funciones `async`, por lo que es muy probable que en poco tiempo comencemos a utilizarla sin tener que incluir el `async` en las funciones. Sin embargo, sólo funcionará en determinados contextos.