Promesas y async wait en Javascript

DWEC – 2º DAW

¿Qué es una promesa?

- Una promesa es un objeto que representa un **valor futuro**, pero no tenemos certeza de cuánto va a tardar y cuándo va a terminar la operación.
- Nos proporciona **una API**, para que cuando la operación termine, poder:
  - Obtener el **valor resultante** o bien
  - Manejar la **excepción** en caso de que se produzca un error.

PROMETO UN RESULTADO!

- El "código productor" es un código que puede llevar algún tiempo (al que esperamos)
- El "código consumidor" es un código que debe esperar el resultado (lo que haremos cuando llegue el resultado)

Una promesa es un objeto de JavaScript que vincula a ambos códigos.

Sintaxis de un objeto promesa

- Creamos objetos promesa con ***new Promise*** con dos parámetros:
  - ***Callback resolve*** --> lo llamaremos cuando la operación se procese correctamente.
  - ***Callback reject*** --> lo usaremos cuando se produzca un error o cuando nosotros consideremos que se ha producido un error

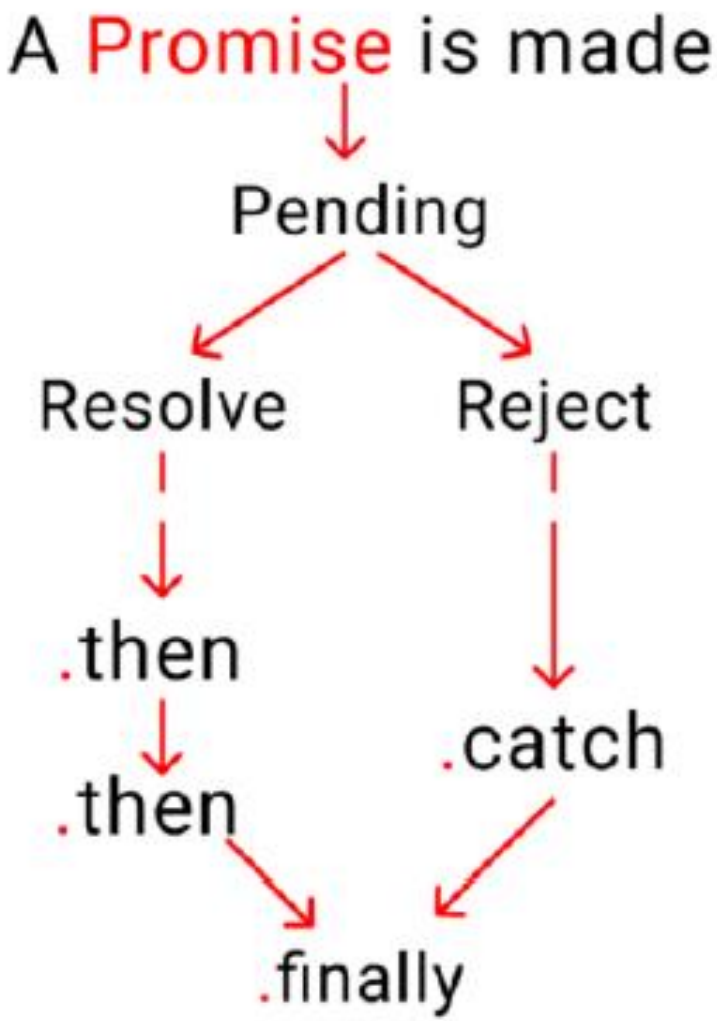
Sintaxis de un objeto Promesa

```
let miPromesa = new Promise(function(miResolve, miReject) {
// "Producing Code" (podría tardar algún tiempo)

  miResolve(); // si éxito
  miReject();  // si error
});

// "Consuming Code" (espera a que se cumpla una promesa)
miPromesa.then(
  function(valor) { /* código si éxito */ },
  function(error) { /* código si error */ }
);
```

Por tanto, cuando el código productor obtiene el resultado, debería llamar a uno de estos callbacks:

Result	Call
Success	myResolve(result value)
Error	myReject(error object)

Esquema de una promesa



Estados de una promesa. Propiedades

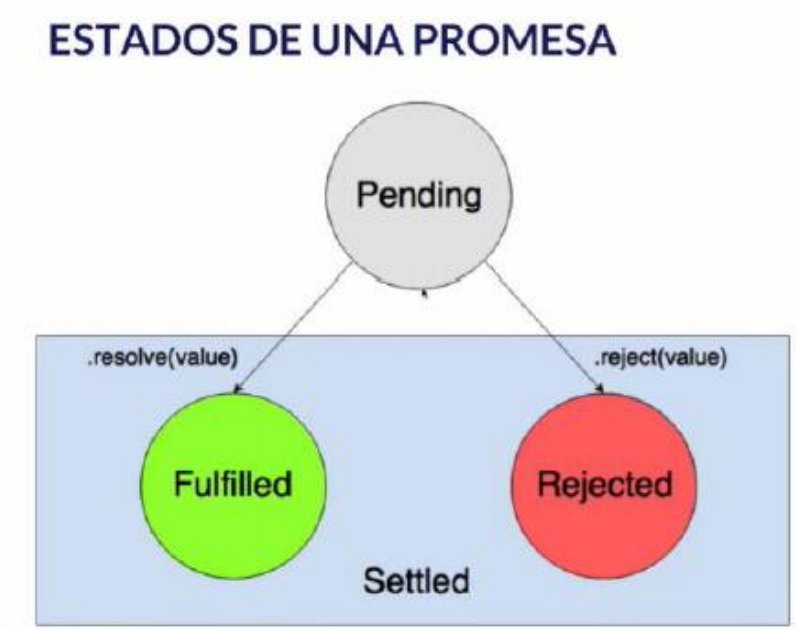
- Un objeto Promise puede encontrarse en **estado**:
  - **Pendiente** (“pending”) / **Resuelto** (“fulfilled”) / **Rechazado** (“rejected”)
- El objeto Promise admite dos propiedades: ***status*** y ***result***.
  - Mientras una promesa está "pendiente", el resultado no está definido.
  - Cuando una Promesa se "cumple", el resultado es un valor.
  - Cuando se "rechaza" una promesa, el resultado es un objeto error.

myPromise.state	myPromise.result
"pending"	undefined
"fulfilled"	a result value
"rejected"	an error object

OJO: No puedes acceder al estado y resultado de las propiedades de Promise. Debes usar un método Promise para manejar las promesas.

Estados de una promesa

Por tanto, una promesa tiene tres estados: **Pendiente, resuelto y rechazado**.



- Si la promesa termina con **resolve()** se llamará a la **primera** función pasada al método **.then()**.
- Si la promesa termina con **reject()** se llamará a la **segunda** función pasada al método .then().
- El método **.catch()** es otra forma alternativa de indicar la **segunda función del .then()**.
- El método pasado a **.finally()** se ejecutaría tanto si la promesa acaba con **resolve()** como si acaba con **reject()**.

Los métodos .then() .catch() .finally()

Métodos	Descripción
.then(resolve)	Ejecuta la función callback resolve cuando la promesa se cumple.
.catch(reject)	Ejecuta la función callback reject cuando la promesa se rechaza.
.then(resolve,reject)	Método equivalente a las dos anteriores en el mismo .then().
.finally(end)	Ejecuta la función callback end tanto si se cumple como si se rechaza.

Con las promesas evitamos el “callback hell” o “infierno de la retrollamada”

https://codearmy.co/que-es-el-callback-hell-y-como-evitarlo-4af418a6ed14

CALLBACK HELL!!

```
hazAlgo(function(resultado) {
    hazAlgoMas(resultado, function(nuevoResultado) {
      hazLaTerceraCosa(nuevoResultado, function(resultadoFinal) {
        console.log('Obtenido el resultado final: ' + resultadoFinal
      }, falloCallback);
    }, falloCallback);
  }, falloCallback);
```

Más sencillo, encadenando promesas. Mostramos el ejemplo con funciones flecha

```
hazAlgo()
.then(resultado => hazAlgoMas(resultado))
.then(nuevoResultado => hazLaTerceraCosa(nuevoResultado))
.then(resultadoFinal => {
  console.log(`Obtenido el resultado final: ${resultadoFinal}`);
})
.catch(falloCallback);
```

Pero aún así tenemos limitaciones…

Enlaces útiles para aprender promesas

- https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Using_promises
- https://www.w3schools.com/js/js_promise.asp
- https://www.freecodecamp.org/espanol/news/promesas-en-javascript-es6/
- https://es.javascript.info/promise-basics (3 tareas interesantes al final)

Ejercicio 1 – parte 1

- La función *setTimeout* utiliza callbacks. Crea una alternativa basada en promesas.
- La función ***delay(ms)*** debería devolver una promesa. Esa promesa debería resolverse después de ***ms*** milisegundos, para que podamos agregarle ***.then***, así:

```
function delay(ms) {
  // tu código
}

delay(3000).then(() => alert('se ejecuta después de 3 segundos'));
```

Ejercicio 1 – parte2

- Dado el siguiente código, modifícalo para que devuelva una promesa y pruébalo.

```
setTimeout(function() {
    escribe("He esperado 4 segundos!!!");
}, 4000);

function escribe(texto) {
  document.getElementById("demo").innerHTML = texto;
}
```

Ejercicio 2

- Crear una promesa para que transcurridos 3 segundos cambie el color de fondo de una web.

Ejercicio 3

- Crear una promesa que en el “resolve” añada la fecha actual a una variable en el localStorage y en el “reject” muestre un mensaje de error.
- ¿Es una promesa segura, es decir, que siempre se va a ejecutar?

ASYNC Y AWAIT

Introducido en ES7

async y *await* nos permiten escribir promesas de forma más sencilla

- **async** hace que una función devuelva una Promise
- **await** hace que una función espera a una Promise

Sintaxis async

- La palabra clave ***async*** antes de una función hace que la función devuelva una promesa:

```
async function myFunction() {
    return "Hello";
}
```

Equivalente a:

```
function myFunction() {
  return Promise.resolve("Hello");
}
```

La consumimos con:

```
myFunction().then(
    function(value) { /* código si exito*/ },
    function(error) { /* código si error*/ }
    );
```

Sintaxis await

- La palabra clave ***await*** solo se puede usar dentro de una función asíncrona (async).
- Hace que la función pause la ejecución y espere una promesa resuelta antes de continuar:

```
let value = await promise;
```

Sintaxis básica async - wait

```
async function myDisplay() {
    let myPromise = new Promise(function(resolve, reject) {
        resolve("I love You !!");
    });
    document.getElementById("demo").innerHTML = await myPromise;
}

myDisplay();
```