Promesas

DigitalHouse >
Coding School

Índice

“

Las **promesas** son funciones que permiten ejecutar código **asincrónico** de forma eficiente.

”

1 Pedidos asincrónicos

Pedidos asincrónicos

Un pedido asincrónico es un conjunto de instrucciones que se ejecutan mediante un mecanismo específico, como por ejemplo un **callback**, una **promesa** o un **evento**. Esto hace posible que la respuesta sea procesada en otro momento.

Como se puede inferir, su comportamiento es **no bloqueante** ya que el pedido se ejecuta en paralelo con el resto del código.

2 | .then()

.then()

La función asincrónica devolverá un resultado, o no. Mientras tanto, el código se **sigue ejecutando**.

{}

```
obtenerUsuarios()

    .then(function(data){
        console.log(data);
    });

console.log("Se sigue ejecutando!")
```

Función asincrónica.

Código que podría seguirse ejecutando mientras se ejecuta la promesa.

.then()

```
{}
```

```
obtenerUsuarios()

    .then(function(data){
        console.log(data);
    });

console.log("Se sigue ejecutando!")
```

Ejecuta el console.log() SOLO SI **obtenerUsuarios()** devuelve un resultado. Este lo recibe **.then()** dentro de su callback, en este caso, en el parámetro **data**.

3 | Pedidos anidados

Pedidos anidados

A veces los `.then()` suelen tener promesas dentro. Para resolver esto, necesitamos utilizar otro `.then()` que entre en ejecución una vez se resuelva el anterior.

```
{}
obtenerUsuarios()
    .then(function(data){
        return filtrarDatos(data);
    })
    .then(function(dataFiltrada){
        console.log(dataFiltrada);
    })
```

¡ATENCIÓN!

Es importante recordar que los **.then()** necesitan retornar la data procesada para que pueda ser usada por otro **.then()**.

.catch()

En caso de **NO** obtener un resultado, se genera un error. Para esto usamos `.catch()`, que encapsula cualquier error que pueda generarse a través de las promesas. Dentro de este método decidimos qué hacer con el error. El mismo es recibido como parámetro dentro del callback del `.catch()`.

En el siguiente ejemplo mostraremos el error en consola:

{}

```
obtenerUsuarios()
    .then(function(data){
        console.log(data);
    })
    .catch(function(error){
        console.log(error);
    })
```

4 | Promise.all()

Promise.all( )

A veces necesitamos que dos o más promesas se resuelvan para realizar cierta acción. Para esto usamos `Promise.all()`. Este contendrá un array de promesas que, una vez se hayan resuelto, se ejecutará un `.then()` con los resultados de las mismas.

Lo que primero debemos hacer es guardar en variables las promesas que necesitamos obtener.

{}

```
let promesaPeliculas = obtenerPeliculas();
```

Promesa de películas

```
let promesaGeneros = obtenerGeneros();
```

Promesa de géneros

Promise.all()

El próximo paso es utilizar el método `Promise.all()` que contendrá un array con las promesas que guardamos anteriormente.

```
Promise.all([promesaPeliculas, promesaGeneros])
```

Promesas a resolver

```
{}
```

Promise.all()

El callback del `.then()` recibe un array con los resultados de las promesas cumplidas.

{}

```
Promise.all([promesaPeliculas, promesaGeneros])

.then(function([resultadoPeliculas,resultadoGeneros
]){
  console.log(resultadoPeliculas,
resultadoGeneros);
})
```

El **.then()** se ejecutará solo si ambas promesas se cumplieron.

Documentación

Para saber más sobre promesas y Promise.all(), podemos acceder a la documentación oficial de Mozilla haciendo click en los siguientes links:

Uso de promesas

Promise.all()


DigitalHouse>
Coding School