



Observable vs. Promesa 9/23



RECURSOS

MARCADORES

JavaScript posee dos maneras de manejar la asincronicidad, a través de: **Observables** y **Promesas**, que comparten el mismo objetivo, pero con características y comportamientos diferentes.

¿Qué es la asincronicidad en JavaScript?

La **asincronicidad** se refiere a cuando Javascript utiliza procesos asíncronos para realizar muchas tareas a la vez, tareas que pueden tomar determinado tiempo o nunca finalizar. Es decir, este lenguaje de programación es un monohilo y esto significa que solo puede hacer una cosa a la vez y la ejecución de un proceso demorará a los que vengan posteriormente hasta que este termine.

Es así como la lectura de archivos o las peticiones HTTP son procesos asíncronos y se requiere de un método para manipular este tipo de procesos como los observables y promesas.

¿Qué son los observables?

Gran parte del ecosistema Angular está basado en observables y la librería **RxJS** es tu mejor aliado a la hora de manipularlos. El patrón de diseño “observador” centraliza la tarea de informar un cambio de estado de un determinado dato o la finalización de un proceso, notificando a múltiples interesados cuando esto sucede sin necesidad de que tengan que consultar cambios activamente.

Características de los Observables en Javascript

- Emiten múltiples datos
- Permiten escuchar cualquier tipo de proceso, (peticiones a una API, lectura de archivos, etc.)
- Notifican a múltiples interesados
- Pueden cancelarse
- Manipulan otros datos (transformar, filtrar, etc.) con **RxJS**.
- Son propensos al callback hell

Ejemplos con Observables

```
import { Observable } from 'rxjs';

const getAnObservable$ = () => {
  return new Observable(observer => {
    observer.next('Valor 1');
    observer.next('Valor 2');
    observer.next('Valor 3');
  });
};

(() => {
  getAnObservable$
    .pipe(
      // Manipulación de resultados con RxJS
    )
    .subscribe(res => {
      console.log(res);
    });
})();
```

¿Qué son las promesas?

Las **promesas** son un método algo más sencillo y directo para manipular procesos asíncronos en Javascript. Además, estos objetos tienen dos posibles estados:

- Resuelto
- Rechazado

Dependiendo si el proceso asíncrono se ejecutó correctamente hubo algún error.

Desde el año 2017 se especificó en el estandar de EcmaScript la posibilidad de manipular promesas de una manera mucho más fácil con **async/await**. *Async* para especificar que una función es asíncrona y *Await* para esperar por el resultado sin bloquear el hilo de ejecución.

Características de las Promesas

- Ofrecen mayor simplicidad
- Emiten un único valor
- Evitan el *callback hell*
- No se puede cancelar
- Proveen una robusta API nativa de Javascript disponible desde ES 2017
- Constituyen librerías populares como **AXIOS** o **Fetch**

Ejemplos con Promesas

```
// Promesas con .then() y .catch()
const p = new Promise((resolve, reject) => {
  setTimeout(function(){
```

```

        resolve("¡Hola Promesa!");
    }, 1000);
});
p.then((result: string) => {
    console.log(result);          // ¡Hola Promesa!
}).catch(err => {
    console.log(err);            // En caso de error
});

// Promesas con async/await
(async () => {
    const p = await new Promise((resolve, reject) => {
        setTimeout(function(){
            resolve("¡Hola Promesa!");
        }, 1000);
    }).catch(err => {
        console.log(err);        // En caso de error
    });
    console.log(p);              // ¡Hola Promesa!
});

```

Observable a Promesa

Una característica más de **RxJS** es la posibilidad de convertir fácilmente un **Observable a Promesa**:

```

import { of, firstValueFrom, lastValueFrom } from 'rxjs';

observableToPromise(): Promise<string> {

```

```
    return lastValueFrom(of('¡Soy una promesa!'));  
  }  
}
```

La función `of` devuelve en forma de observable lo que sea que le coloques dentro. La función `firstValueFrom` o `lastValueFrom` devuelve el primer (o último) valor que el observable emita en forma de promesa.

Promesa a Observable

De manera muy similar, puedes convertir una Promesa en un Observable:

```
import { from } from 'rxjs';  
  
PromiseToObservable(): Promise<Observable<any>> {  
  return from(new Promise((resolve, reject) => { console.log('¡Soy un observable!')  
})));  
}
```

















La función `from` de **RxJS** convertirá una promesa en observable para que puedas manipular y suscribirte a la emisión de sus datos.

Conclusión

En ocasiones te sentirás mejor trabajando de las dos maneras, tanto con el observable como con la promesa. Lo importante es comprender cómo funcionan ambos objetos, sus características, diferencias y decidir cuál aplicar en tus proyectos de programación.


Contribución creada por: Kevin Fiorentino.

Archivos de la clase

 /	
 tsconfig.app.json	
 package.json	
 angular.json	
 .eslintrc.json	
 package-lock.json	
 .browserslistrc	
 .gitignore	
 obs-promises.js	
 tsconfig.spec.json	
 tsconfig.json	
 README.md	

 .editorconfig



 karma.conf.js



 src

Lecturas recomendadas



Angular

<https://angular.io/guide/observables-in-angular>



Angular

<https://angular.io/guide/comparing-observables>

