Promesas en grupo (Promise API) - Javascript en español

Ahora que sabemos ¿Qué son las promesas?, para qué y como se usan, podemos profundizar y aprender más sobre la **API Promise** nativa de Javascript, mediante la cuál podemos realizar operaciones con grupos de promesas, tanto independientes como dependientes entre sí.

> Por norma general, las tareas asíncronas no sabemos cuanto tardarán en responder y/o procesarse, por lo que muchas veces el orden en que se resuelvan no será el mismo. Esto en algunos casos no nos importará, pero en otros sí, por lo que hay que tenerlo en cuenta.

API de las promesas

El objeto `Promise` de Javascript tiene varios métodos estáticos que podemos utilizar en nuestro código. Todos devuelven una promesa y son los que veremos en la siguiente tabla:

Métodos	Descripción
`Promise.all(list)`	Acepta sólo si **todas** las promesas del se cumplen.
`Promise.allSettled(list)`	Acepta sólo si **todas** las promesas del se cumplen o rechazan.
`Promise.any(value)`	Acepta con el valor de la primera promesa del que se cumpla.
`Promise.race(value)`	Acepta o rechaza dependiendo de la primera promesa que se procese.
`Promise.resolve(value)`	Devuelve un valor envuelto en una promesa que se cumple directamente.
`Promise.reject(value)`	Devuelve un valor envuelto en una promesa que se rechaza directamente.

En los siguientes ejemplos, vamos a utilizar la función `fetch()` para realizar varias peticiones y descargar varios archivos diferentes que necesitaremos para nuestras tareas.

Promise.all()

El método `Promises.all()` funciona como un «**todo o nada**»: devuelve una promesa que se cumple cuando todas las promesas del se cumplen. Si alguna de ellas se rechaza, `Promise.all()` también lo hace.

En nuestro ejemplo, cada uno de los `fetch()` tendrá su propia promesa y **sólo cuando se hayan descargado los tres archivos de cada petición** se cumplirá la promesa del `Promise.all()`:

```
const p1 = fetch("/robots.txt");
```

```
const p2 = fetch("/index.css");
const p3 = fetch("/index.js");

Promise.all([p1, p2, p3])
  .then(responses => {
    responses.forEach(response => {
      console.log(response.status, response.url);
    })
  });
```

A `Promise.all()` le pasamos un con las promesas individuales. Cuando **todas y cada una** de esas promesas se cumplan favorablemente, **entonces** se ejecutará la función callback de su `.then()`. En el caso de que alguna se rechace, no se llegará a ejecutar.

Promise.allSettled()

El método `Promises.allSettled()` funciona como un «**todas procesadas**»: devuelve una promesa que se cumple cuando todas las promesas del se hayan procesado, independientemente de que se hayan cumplido o rechazado.

```
const p1 = fetch("/robots.txt");
const p2 = fetch("https://google.com/index.css");
const p3 = fetch("/index.js");

Promise.allSettled([p1, p2, p3])
  .then(responses => {
    responses.forEach(response => {
      console.log(response.status, response);
    })
  });
```

Esta promesa nos devuelve un campo `status` donde nos indica si cada promesa individual ha sido cumplida o rechazada, y un campo `value` con los valores devueltos por la promesa. En este caso, obtendremos que la primera y última promesa se resuelven (*fulfilled*), mientras que la segunda nos da un error de CORS y se rechaza (*rejected*).

Promise.any()

El método `Promise.any()` funciona como «**la primera que se cumpla**»: Devuelve una promesa con el valor de la primera promesa individual del que se cumpla. Si todas las promesas se rechazan, entonces devuelve una promesa rechazada.

```
const p1 = fetch("/robots.txt");
const p2 = fetch("/index.css");
const p3 = fetch("/index.js");

Promise.any([p1, p2, p3])
  .then(response => console.log(response.status, response.url));
```

Como vemos, `Promise.any()` devolverá una respuesta de la primera promesa cumplida.

Promise.race()

El método `Promise.race()` funciona como una «**la primera que se procese**»: la primera promesa del que sea procesada, independientemente de que se haya cumplido o rechazado, determinará la devolución de la promesa del `Promise.race()`. Si se cumple, devuelve una promesa cumplida, en caso negativo,

devuelve una rechazada.

```
const p1 = fetch("/robots.txt");
const p2 = fetch("/index.css");
const p3 = fetch("/index.js");

Promise.race([p1, p2, p3])
  .then(response => console.log(response.status, response.url));
```

De forma muy similar a la anterior, `Promise.race()` devolverá la promesa que se resuelva primero, ya sea cumpliéndose o rechazándose.

Promise.resolve() y Promise.reject()

Mediante los métodos estáticos `Promise.resolve()` y `Promise.reject()` podemos devolver una promesa cumplida o rechazada respectivamente sin necesidad de crear una promesa con `new Promise()`. Esto puede ser interesante en algunos casos, aunque rara vez solemos utilizarlo hoy en día.

```
const doTask = () => {
  const number = 1 + Math.floor(Math.random() * 6);
  return (number % 2 === 0) ? Promise.resolve(number) : Promise.reject(number);
}
```

Observa que en este caso devolvemos una promesa que se cumple cuando el número generado es par y se rechaza cuando es impar. Sin embargo, ten en cuenta que en problema en este caso es que la promesa no «envuelve» toda la función, por lo que si la tarea tardase algún tiempo en generar el número, no podríamos utilizar el `.then()` para consumir la promesa.

> Estas funciones estáticas se suelen utilizar en muy pocos casos, para mantener cierta compatibilidad en funciones que se espera que devuelvan una promesa.