Promesas múltiples y promesas anidadas

Promesas múltiples

¿En que caso usamos promesas múltiples?

Un ejemplo de aplicación, es cuando requerimos distintas llamadas asincrónicas a APIs, y es necesaria la respuesta de todas las partes para poder construir un objeto de respuesta.

Entonces, partimos de que las promesas tienen sólo 3 estados posibles: pending, resolve, rejected.

¿Cómo se usa?

Promise.all(parametro);

parámetro es un arreglo, sobre el cual se podrá iterar.

¿Qué devuelve?

Una Promise que se cumplirá cuando todas las promesas del argumento *parámetro* hayan sido cumplidas, o bien se rechazará cuando alguna de ellas se rechace.

Importante:

Promise.all se cumple cuando todas las promesas del *parámetro* dado se han cumplido, o es rechazada si alguna promesa no se cumple. ***Promise.resolve***.

Si alguna de las promesas pasadas en el argumento *parámetro* falla, la promesa all es rechazada inmediatamente con el valor de la promesa que fué rechazada, descartando todas las demás promesas hayan sido o no cumplidas. Si se pasa un array vacío a all , la promesa se cumple inmediatamente.

Ejemplo:

```
Promise.all([servicio1, servicio2, servicio3])
.then(function(response){
        console.log(response)
})
.catch(function(error){
})
```

Promesas anidadas

En muchas ocasiones es necesario realizar más de una llamada a una API o acceder a recursos adicionales de manera asincrónica.

Partiendo de la estructura básica:

```
fetch()
.then(function(response){ })
.catch(function(error){
});
```

Fetch, realiza una llamada asincrónica y devuelve una promise.

Ejemplo completo:

```
fetch('http://example.com/movies.json')
  .then(function(response) {
    return response.json();
  })
  .then(function(myJson) {
    console.log(myJson);
  });
```

Siempre, que tengamos una llamada asincrónica debemos esperar por la respuesta, para poder mostrar los resultados.

Entonces:

Debemos hacer que nuestra función, espere el resultado de la nueva llamada asincrónica y para esto, haremos uso del operador ***wait.***

wait:

El operador await es usado para esperar a una Promise. Sólo puede ser usado dentro de una función async function.

Resumen:

La expresión await provoca que la ejecución de una función async sea pausada hasta que una Promise sea terminada o rechazada, y regresa a la ejecución de la función async después del término. Al regreso de la ejecución, el valor de la expresión await es la regresada por una promesa terminada.
Si la Promise es rechazada, el valor de la expresión await tendrá el valor de rechazo.
Si el valor de la expresión seguida del operador await no es una promesa, será convertido a una resolved Promise.

Ejemplo de implementación

La siguiente función genera un mensaje de bienvenida personalizado, requiriendo el nombre del usuario de una llamada asincrónica a un servicio.

```
async function miFuncion() {
  var saludo = “Bienvenido”;
  var x = await llamoServicio();
  saludo = saludo+x;
  console.log(x);
}
miFuncion();
```

Como lo interpretamos, en nuestro código js, llamamos a *miFuncion(),* esta, utiliza una llamada asincrónica para construir el mensaje, por lo que debe esperar el resultado retornado para luego continuar operando y generar la respuesta final.

La línea:

```
var x = await llamoServicio();
```

Ejecuta la llamada y la expresión ***await,*** nos indica que aguardaremos al resultado de la promesa.

Importante: para que esto funcione tal como lo esperamos, la función que realiza la llamada (miFuncion()), debe ser declarada como asincrónica, utilizando el modificador *async.*

Nota: este tipo de llamadas, serán utilizadas cada vez que sea requerido implementar llamadas asincrónicas, no sólo para resolver promesas anidadas, esta última es una aplicación más simplemente.