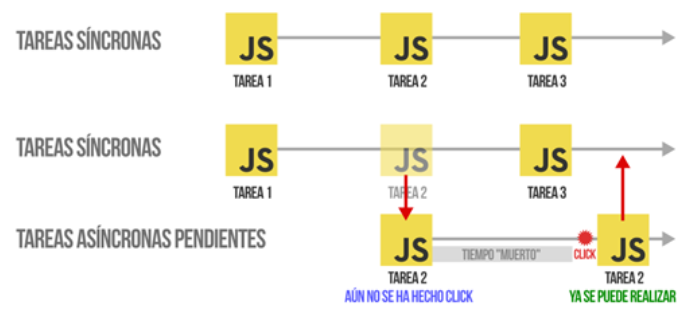
PROGRAMACION ASINCRONA

ASINCRONIA

En un ambiente síncrono solo podemos tener una tarea o proceso ejecutándose a la vez y debemos esperar a que termine para ejecutar el siguiente proceso. Esto significa que debemos hacer una operación primero, esperar a que termine e iniciar la segunda operación. La desventaja es que el tiempo total será la suma del tiempo de respuesta de ambas operaciones, además que bloqueamos cualquier otro proceso. En un sistema síncrono la solución a este problema sería iniciar otro thread.

Un thread es un programa corriendo cuya ejecución se puede intercalar con otros programas del sistema operativo, esto es posible porque la mayoría de computadoras modernas cuentan con múltiples procesadores.



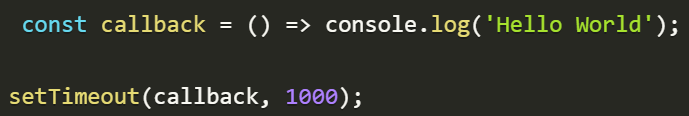
JavaScript es un lenguaje de programación de un solo thread, es decir, no podemos usar varios procesadores. El motor de JavaScript solo puede procesar una sentencia a la vez en un único thread. Si estamos haciendo una operación como las descritas anteriormente nuestro thread principal se bloquea hasta que dicha operación termine.

Aquí es donde JavaScript asíncrono entra en juego. Mediante el uso de callbacks, promesas y async/await podemos realizar operaciones con largos tiempos de respuesta sin bloquear el thread principal.

CALLBACKS

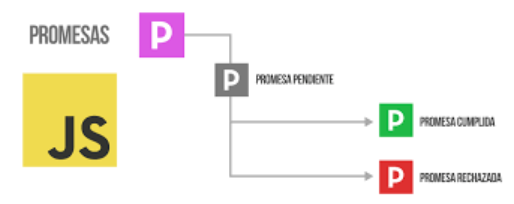
Un callback es una función que se pasa como argumento a otra función para que sea ejecutada al final de algún proceso. El callback también puede ser asignado a una variable para pasarlo a una función.



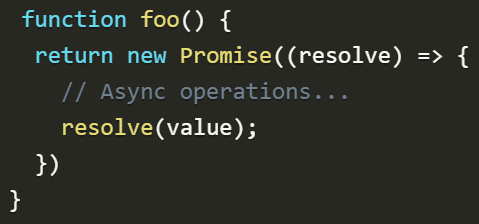


PROMESAS

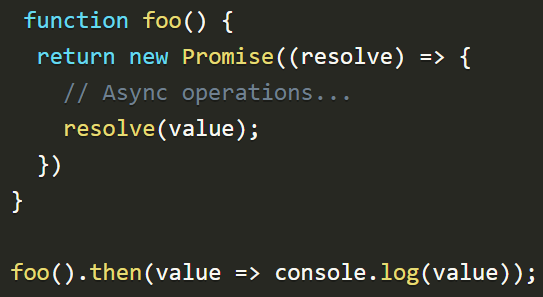
Una promesa es una acción asíncrona que podría cumplirse en algún momento y producir un valor. En cierta forma es igual a los callbacks pero con una mejor sintaxis que facilita la legibilidad.



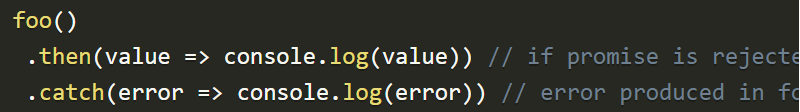
Para crear promesas usamos el constructor Promise el cual espera una función como argumento (se ejecuta de inmediato) que a su vez recibe una función usada para resolver la promesa.



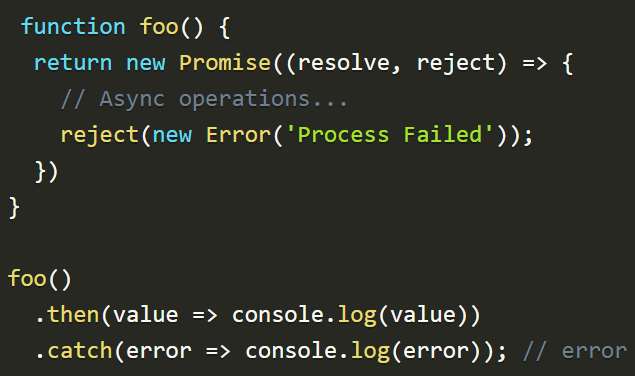
Podemos obtener el resultado de una promesa usando el método then, éste registra una función callback para ser llamada cuando la promesa se resuelve y produce un valor.



En ocasiones las operaciones asíncronas pueden llegar a fallar. Para manejar esto podemos usar el método catch el cual registra un callback a ejecutar cuando una promesa no se resuelve, similar a la forma en que trabaja then. Cuando una promesa es rechazada la promesa que produce then también es rechazada, esto significa que si tenemos una cadena de acciones asíncronas y uno de los pasos llega a fallar, toda la cadena se rechaza.



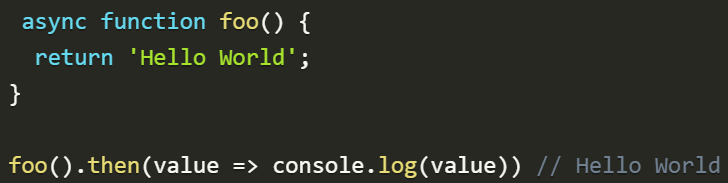
La función que pasamos como argumento al constructor Promise recibe un segundo argumento que podemos utilizar para rechazar una promesa.



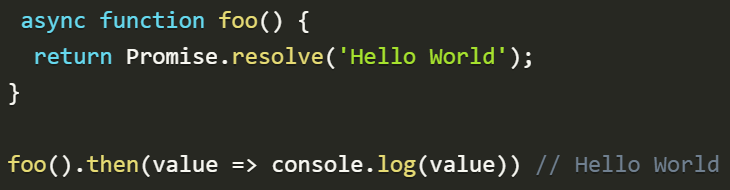
ASYNC / AWAIT

Una de las características de ES6 es async/await. La finalidad de este tipo de funciones es simplificar el comportamiento y uso de promesas.

Al colocar el keyword async en la declaración de una función, estamos definiendo una función asíncrona. Esta función siempre retornará una promesa, esto significa que si el valor retornado no es una promesa automáticamente se envuelve dentro de una.



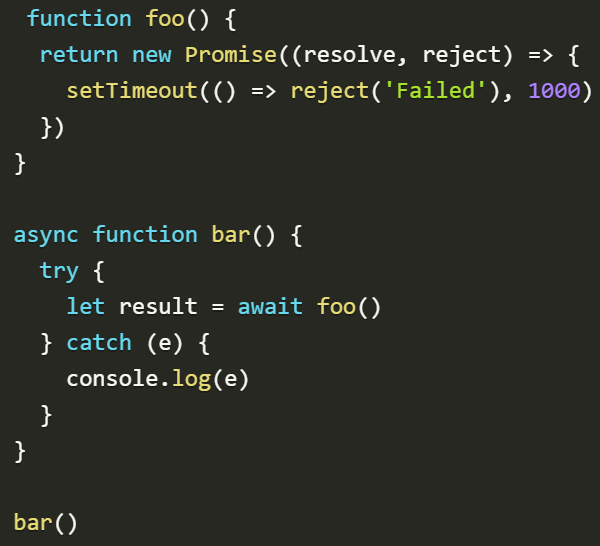
También podemos retornar explícitamente una promesa y el resultado será el mismo.



El keyword await solo puede ser usado dentro de funciones async nos permite esperar hasta que una promesa sea resuelta y retorne un valor.

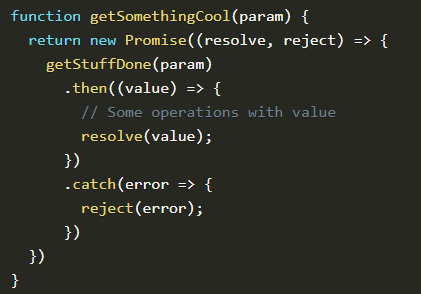


La función pausa su ejecución cuando encuentra un await y continua una vez que la promesa haya sido resuelta. Debemos envolver esta operación en un try...catch en caso que la promesa sea rechazada.



ANTIPATRON

A esto se le conoce como un anti patrón, el problema con getSomethingCool está en la promesa dentro del constructor Promise. Si getSuffDone llega a lanzar una excepción el error pasa desapercibido en lugar de propagarse a la siguiente promesa, como resultado nos quedamos con una promesa en estado pending de manera indefinida.



Las promesas además de encadenarse con .then se pueden retornar directamente, por lo que podemos reescribir la función para no usar new Promise y así propagar los errores de manera correcta. El mismo error se puede cometer trabajando con async/await.

