Contenido

Introducción asincronía 2
Temporizadores: 2
setTimeout() 2
setInterval 2
Single thread y Multi thread 3
Concurrencia y Paralelismo 3
Concurrencia 3
Paralelismo 4
Bloqueante y No Bloqueante 4
Bloqueante 4
No Bloqueante 4
Síncrono y Asíncrono 4
Síncrono 4
Asíncrono 4
Ejemplo Síncrono 5
Ejemplo Asíncrono 5
Event Loop 6
Mecanismos asíncronos 7
Callbacks 7
Promises 7
Async / Await 7
Recursos 9

Introducción asincronía

La asincronía es uno de los pilares fundamentales de Javascript, ya que es un lenguaje de programación de un sólo subproceso o hilo (single thread), lo que significa que sólo puede ejecutar una cosa a la vez.

Si bien los idiomas de un sólo hilo simplifican la escritura de código porque no tiene que preocuparse por los problemas de concurrencia, esto también significa que no puede realizar operaciones largas como el acceso a la red sin bloquear el hilo principal.

Imagina que solicitas datos de una API. Dependiendo de la situación, el servidor puede tardar un tiempo en procesar la solicitud mientras bloquea el hilo principal y hace que la página web no responda.

Ahí es donde entra en juego la asincronía que permite realizar largas solicitudes de red sin bloquear el hilo principal.

JavaScript fue diseñado para ser ejecutado en navegadores, trabajar con peticiones sobre la red y procesar las interacciones de usuario, al tiempo que mantiene una interfaz fluida.

Javascript usa un modelo asíncrono y no bloqueante, con un loop de eventos implementado en un sólo hilo, (single thread) para operaciones de entrada y salida (input/output).

Gracias a esta solución, Javascript es áltamente concurrente a pesar de emplear un sólo hilo.

Temporizadores:

setTimeout()

es usada para retrasar la ejecución de la función pasada como argumento por un periodo de tiempo determinado.

```
setTimeout(function(){
    console.log("Hola Mundo");
}, 2000);


console.log("setTimeout() Ejemplo...");
```

setInterval

Usa setInterval() para hacer que una función se repita con un tiempo de retraso entre cada ejecución.

```
setInteval(function(){
    console.log("Hola Mundo");
},  2000);

console.log("setTimeout() Ejemplo...");
```

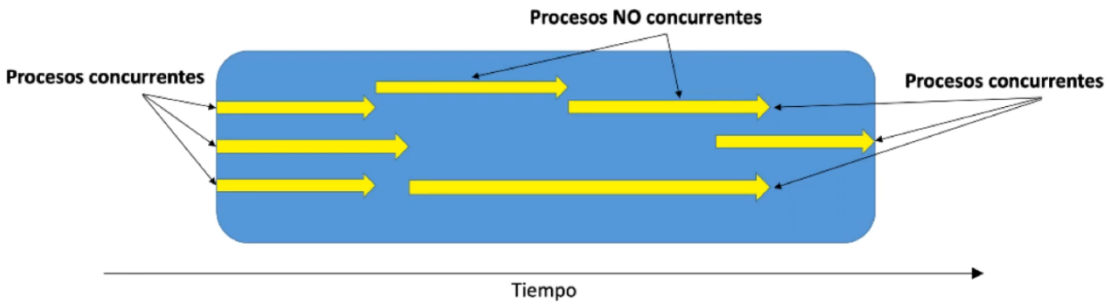
Single thread y Multi thread

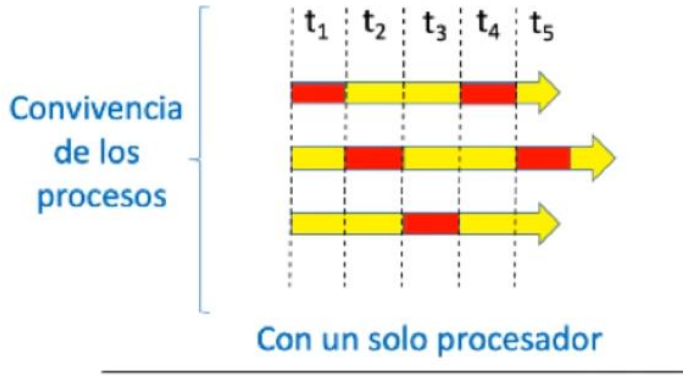
Un hilo la unidad básica de ejecución de un proceso, cada que abres un programa como el navegador o tu editor de código, se levanta un proceso en tu computadora e internamente este puede tener uno o varios hilos (threads) ejecutándose para que el proceso funcione.

Concurrencia y Paralelismo

Concurrencia

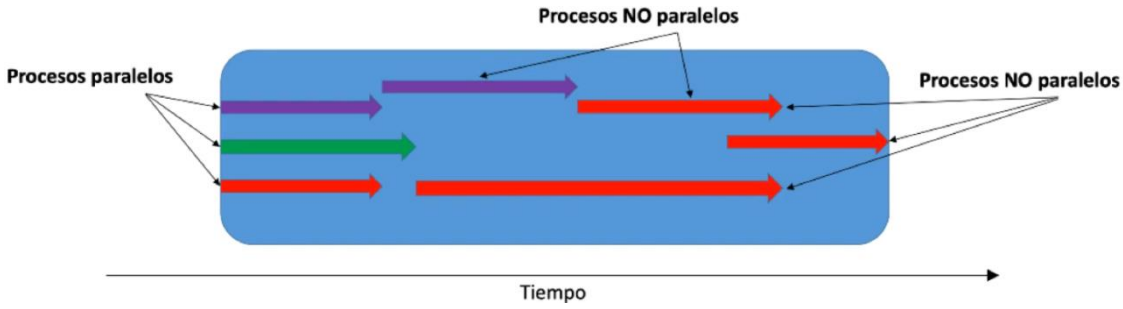
La concurrencia es la capacidad de un sistema para procesar más de un hilo de ejecución (proceso) al mismo tiempo.





Paralelismo

cuando dos o más tareas se ejecutan, al mismo tiempo.



Bloqueante y No Bloqueante

Se refiere a como la fase de espera de las operaciones afectan a nuestra aplicación:

Bloqueante

Son operaciones que no devuelven el control a nuestra aplicación hasta que se ha completado. Por tanto, el thread queda bloqueado en estado de espera.

No Bloqueante

Son operaciones que devuelven inmediatamente el control a nuestra aplicación, independientemente del resultado de esta. En caso de que se haya completado, devolverá los datos solicitados. En caso contrario (si la operación no ha podido ser satisfecha) podría devolver un código de error.

Síncrono y Asíncrono

Se refiere a ¿cuándo tendrá lugar la respuesta?:

Síncrono

La respuesta sucede en el presente, una operación síncrona esperará el resultado.

Asíncrono

La respuesta sucede a futuro, una operación asíncrona no esperará el resultado.

Con lo anterior en JavaScript podemos tener:

Código síncrono y bloqueante o

Código asíncrono y no bloqueante

Ejemplo Síncrono

```
console.log("Inicio");

function dos() {
 console.log("Dos");
}

function uno() {
 console.log("Uno");
 dos();
 console.log("Tres");
}

uno();
console.log("Fin");
```

El resultado en consola es:

```
Inicio
Uno
Dos
Tres
Fin
```

Ejemplo Asíncrono

```
console.log("Inicio");

function dos() {
 setTimeout(function () {
  console.log("Dos");
 }, 1000);
}

function uno() {
 setTimeout(function () {
  console.log("Uno");
 }, 0);
 dos();
 console.log("Tres");
```

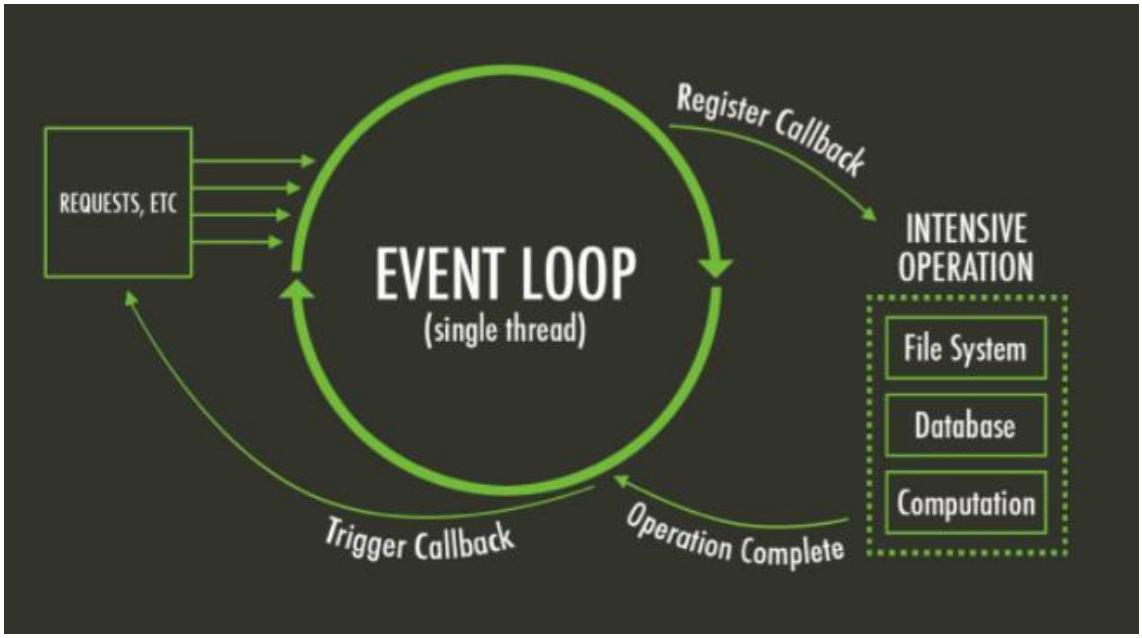
```
}

uno();
console.log("Fin");
```

El resultado en consola es:

```
Inicio
Tres
Fin
Uno
Dos
```

Event Loop

Es una especie de "watcher " y observador que mueve las tareas del callback queue hacia el call stack una vez que esta vacio para que sean ejecutadas.



Mecanismos asíncronos

- Callbacks.
- Promises.
- Async / Await

Callbacks

Una callback es una función que se ejecutará después de que otra lo haga.

Promises

Una promesa es un objeto que representa el resultado de una operación asíncrona y tiene 3 estados posibles:

- Pendiente.
- Resuelta.
- Rechazada.

Tienen la particularidad de que se pueden encadenar (then), siendo el resultado de una promesa, los datos de entrada de otra posible función.

Las promesas mantienen un código más legible y mantenible que las callbacks, además tienen un mecanismo para la detección de errores (catch) que es posible usar en cualquier parte del flujo de datos.

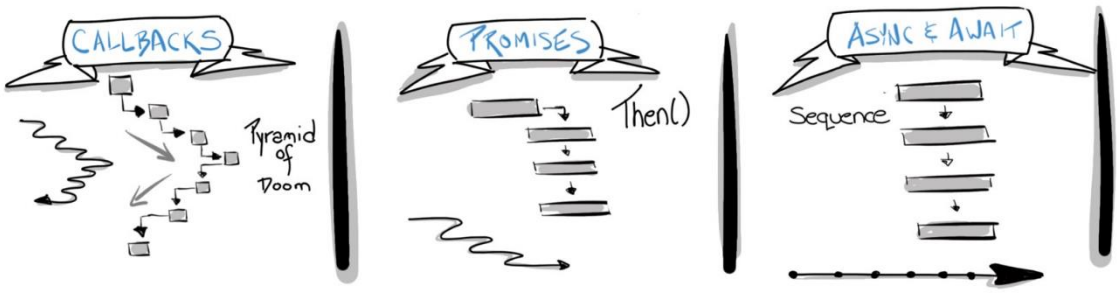
Async / Await

Las funciones asíncronas (async / await) surgen para simplificar el manejo de las promesas.

La palabra async declara una función como asíncrona e indica que una promesa será automáticamente devuelta.

La palabra await debe ser usado siempre dentro de una función declarada como async y esperará de forma asíncrona y no bloqueante a que una promesa se resuelva o rechace.

Asynchronous TypeScript



Recursos

http://latentflip.com/loupe

https://www.freecodecamp.org/

https://developer.mozilla.org/

https://aprendejavascript.org/

https://www.youtube.com/playlist?list=PLImOJ2OqvvkCuDi6E33HXMP23BvYYBHcm