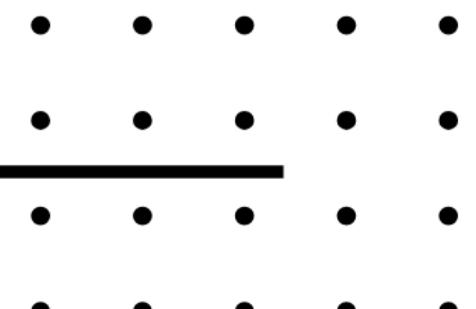


• • • • •

César Alberto Cázares Chaván 82196

T1_ASINCRONISMO



Asincronismo

Introducción

Este es un paradigma de programación que permite a una aplicación ejecutar tareas de larga duración sin bloquear el hilo principal de ejecución. La diferencia con el modelo sincrónico, donde cada tarea debe terminar antes de que comience la siguiente. Esta investigación analizará en la funcionalidad interna del asincronismo.

Su función principal es iniciar el "bloqueo" del hilo de ejecución principal de un programa.

En lugar de forzar a la aplicación a esperar pacientemente a que se completen tareas lentas, que permite que el programa delegue estas tareas. Mientras espera la respuesta, el programa puede continuar ejecutando otro trabajo.

La función de este se basa en gestionar tareas concurrentes a través de un solo hilo. El componente central que la hace posible en entornos como JavaScript (Node.js) es el bucle de Eventos o event loop.

- Su proceso funciona así:
- Call stack: donde se ejecuta el código principal
- Operaciones Asincronas: cuando el Call Stack encuentra otra operación asincrona
- Hilo sigue: inmediatamente después de delegar, el call stack queda libre
- Callback Queue: cuando la operación externa termina
- Event loop: el maestro

Asincronismo en métodos (async/await)

Con el propósito de facilitar el trabajo con Promesas aún más simple & legible se introdujo el async/await

async: palabra clave que se pone antes de una función que devuelva una promesa
await: sólo se puede usar dentro de una función async. cuando se coloca antes de esa Promesa, await pausa la ejecución de esa función hasta que la Promesa se resuelva.

Asincronismo en Middleware

En un servidor, tareas de middleware como la autenticación necesitan hacer I/O

Si fueran sincronas, bloquearían todo el servicio.

Al usar `async` y `await` el middleware pone en solo esa petición. Mientras espera la respuesta de la base de datos, el resto principal queda libre para procesar otras peticiones de otras secciones. Esto es crucial para que el servicio sea eficiente y pueda manejar miles de conexiones a la vez.

Ejemplo:

- se declara la función `async e inicial método`.
- después se utiliza `await` para llamar a la función
- Efecto asincrono: En lugar de pausar el programa, `await` solo pausa la ejecución dentro de esa función.
- después de llamar a la función, imprimimos un mensaje
- resultado: inicia el método, se imprime el mensaje. después, la función se reanuda e imprime

Conclusión

El asincronismo es un pilar fundamental para crear aplicaciones modernas, rápidas y escalables ya que evita el bloqueo del flujo de ejecución principal.

En los métodos, la sintaxis `async await` ayuda enormemente en la gestión de operaciones de I/O. En el middleware, este mismo principio es el motor, que permite a los servidores web manejar miles de peticiones simultáneas sin competir, procesando otras solicitudes mientras espera respuestas leídas.

Referencias

- Node.js Documentation (2025). Overview of Blocking vs Non-Blocking APIs. [express.js](#) (2025). Using middleware and writing for use in express apps
- Microsoft Docs. Programación asincrónica con Async y Await (C#)