César Alberto Cázares Chaván 82196

T1_ASINCRONISMO

Universidad La Salle Bajío
Facultad de Ingenierías y Tecnologías

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 a diferencia del modelo síncrono, donde cada tarea debe terminar antes de que comience la siguiente. Esta investigación analizará en la funcionalidad interna del asincronismo.

Su función principal es evitar el "bloqueo" del hilo de ejecución principal de un programa.
En lugar de forzar a la aplicación a esperar ociosamente 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.

En función de este se basa en gestionar tareas concurrentes a menudo en un solo hilo. El componente central que lo 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 Asíncronas: Cuando el Call Stack encuentra una operación asíncrona
- Hilo sigue: inmediatamente después de delegar, el call stack queda libre
- Callback Queue: Cuando la operación externa termina
- Event loop: el mánita

Asincronismo en metodos (async/await)
Con el propósito de facilitar el trabajo con Promesas aún más simple y legible se introdujo el async/await

async: palabra clave que se pone antes de una función devuelve una Promesa
await: Solo se puede usar dentro de una función async. Cuando se coloca antes de una Promesa, await para 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 síncronas, bloquearían todo el servidor.
Al usar async y await el middleware puede salir de esa petición. Mientras espera la respuesta de la base de datos, el hilo principal queda libre para procesar otras peticiones de otras usuarios. Esto es crucial para que el servidor sea eficiente y pueda manejar miles de conexiones a la vez.

ejemplo:
- se declara la función async e inicia el método.
- después se utiliza await para llamar a la tarea
- Efecto asíncrono: En lugar de congelar 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 hilo de ejecución principal.

En los métodos, la sintaxis async await ayuda enormemente, la gestión de operaciones de I/O. En el middleware, este mismo principio es el mata, que permite a los servidores web manejar miles de peticiones simultaneas sin colapsar, procesando otras solicitudes mientras espera respuestas lentas.

Referencias

Node.js Documentation (2025). Overview of blocking vs Non-Blo

Express.js Guide (2025). Using middleware y Writing for use in express apps

Microsoft Docs. Programación asincrónica con Async y Await (C#)