Exposición : Asincronía Web

Sistemas Operativos

Ing. Gunnar Eyal Wolf Iszaevich

Ricardo Rosales Romero

26 Septiembre 2019

Introducción

Asincronía en Javascript

Al desarrollar aplicaciones web nuestro código Front-End se ejecuta bajo un solo Thread(ThreadUI) y las peticiones que tengamos que hacer a servidor o procesos que llevan un largo tiempo de ejecución a menudo tienen unos tiempos de respuesta que podrían bloquear el Thread, inhabilitando así la aplicación durante el proceso y por ende mostrando al usuario final un comportamiento fuera del esperado.

La única forma que hasta el momento se, es liberar ese Thread principal es creando código asíncrono para nuestras llamadas y hasta el momento nos hemos servido de diferentes actores para desempeñar este papel tales como callbacks o bien de forma más óptima encapsulando las respuestas en objetos Promise.

Desarrollo

Callbacks

Definición

Los callbacks son la pieza clave para que la web pueda funcionar de forma asíncrona. Es decir que el resto de patrones asíncronos en Javascript está basado en callbacks de algún modo, simplemente añaden azúcar sintáctico para trabajar con ellos de una forma más comoda.

Un callback no es más que una función que se pasa como argumento de otra función, y que será invocada para completar algún tipo de acción. En nuestro contexto asíncrono, un callback representa el 'Qué quieres hacer una vez que tu operación asíncrona termine?'. Por tanto, es la parte de código que será ejecutada una vez que una operación asíncrona notifique que ha terminado. Esta ejecución se hará en algún momento futuro, gracias al mecanismo que implementa el bucle de eventos.

Ejemplo

```
1 console.log("Yo estoy siendo impreso");
2
3 setTimeout( () => {
4         console.log("Hola como estan ?");
5         } }, 2000)
6
```

```
7 console.log("Hola desde B205");
```

Promesas

Definición

Las Promemas vienen, para hacer legibles los códigos frenéticos de flujos de callbacks anidados en una estructura más robusta y así manejar nuestras callbacks de una manera más eficiente. Además cumple con un concepto similar a la de una máquina de estados porque está preparada para responder a un eventual valor en el futuro, bajo una relación de estados cuya representación con los datos entrantes es poco predecible. En definitiva una Promesa es un objeto que sirve para encapsultar una eventual 'tarea'.

- **Pendiente**
- **Resuelta**
- **Rechazada**

Ejemplo

```
1 const delay = tiempo => new Promise(resolve => setTimeout(
     resolve, tiempo));
2
3 delay(4000)
4   .then(() => console.log(`Este es un retardo de al menos 4
        segundos`))
5   .catch(() => console.log(`Retardo fallido`));
```

Async- await

Definición

Las promesas supusieron un gran salto en Javascript al introducir una mejora sustancial sobre los callbacks y un manejo elegante de la asincronía. Sin embargo, también pueden llegar a ser tediosas y verbosas a medida que se requieren más y más .then(). Las palabras clave async y await surgieron para simplificar el manejo de las promesas. Son el puro azúcar para hacer las

promesas más amigables, escribir código sencillo, reducir el anidamiento y mejorar la trazabilidad al depurar. Pero recuerda, async await y las promesas son lo mismo.

La etiqueta async declara una función como asíncrona e indica que una promesa será automáticamente devuelta. Podemos declarar como async tanto funciones con nombre, anónimas, o funciones flecha. Por otro lado, await debe ser usado siempre dentro de una función declarada como async y va a esperar de forma automática (de forma asíncrona y no bloqueante) a que una promesa se resuelva.

Ejemplo

```
async function wait() {
  await delay(1500);
  await delay(1500);
  return "Ha transcurrido, como minimo, 3 segundos.";
};
```

Event Loop

El event loop se encarga de implementar las operaciones asíncronas o el non-blocking. El event loop corre en el único hilo que existe en Node y como mencionamos anteriormente, al bloquear el único hilo de node, estamos bloqueando el event loop. El event loop es el que se encarga de revisar que el call stack este vacío para añadir lo que está dentro del callback queue y ejecutarlo.

Callback Queue

Aquí se agregan los callback o funciones que se ejecutan una vez las operaciones asíncronas hayan terminado. Se utiliza el método FIFO (first input, first output), traducido, primero en entrar, primero en salir , aunque no siempre aplica.

Cada vez que nuestro programa recibe una notificación del exterior o de otro contexto distinto al de la aplicación (como es el caso de operaciones asíncronas), el mensaje se inserta en una cola de mensajes pendientes y se registra su callback correspondiente. Recordemos que un callback era la función que se ejecutará como respuesta.

Conclusiones

Asincronía

Cuando se abordan temas de control de flujo de cualquier sistemas se tiene que hablar con cautela ya que muchas veces se suelen mezclar definiciones similares, lo que se vio en clase acerca de cómo los programadores resolvieron problemas que se presentaron a a la hora de implementar sistemas operativos que en esencia deben de ser capaces de administrar de forma correcta la forma en la cual se llevan acabo los procesos dentro de un equipo de uso personal o empresarial. Algo así propulsó mi idea de mostrar frente al grupo las formas de resolver esa clase de problemas que también son presentados en esa herramienta indispensable para cada uno de nosotros llamada web . Pasando por varios conceptos patrones y ejemplos , concluí que efectivamente no hay mucho en común y algunos conceptos distan, aún así siempre habrá algo en común por tratarse de una computadora y procesos que deben ejecutarse de forma simultánea o casi simultánea dentro de su entorno.

Referencias

Bibliográficas

- **Javascript Asíncrono: La guía definitiva, Obtenido de : `https://lemoncode.net/lemoncode-blog/2018/1/29/javascript-asincrono`**
- **Asincronía en JavaScript , Obtenido de : `https://itblogsogeti.com/2016/07/27/asincronia-en-javascript/`**
- **Node Js y el Event Loop , Obtenido de : `https://medium.com/@_ferh97/nodejs-y-el-event-loop-21b33fea6b03`**