educación IT
CODE YOUR FUTURE

Bootcamp Java Developer

Fase 2 - Java Web Developer

Módulo 19

Procesos asincrónicos

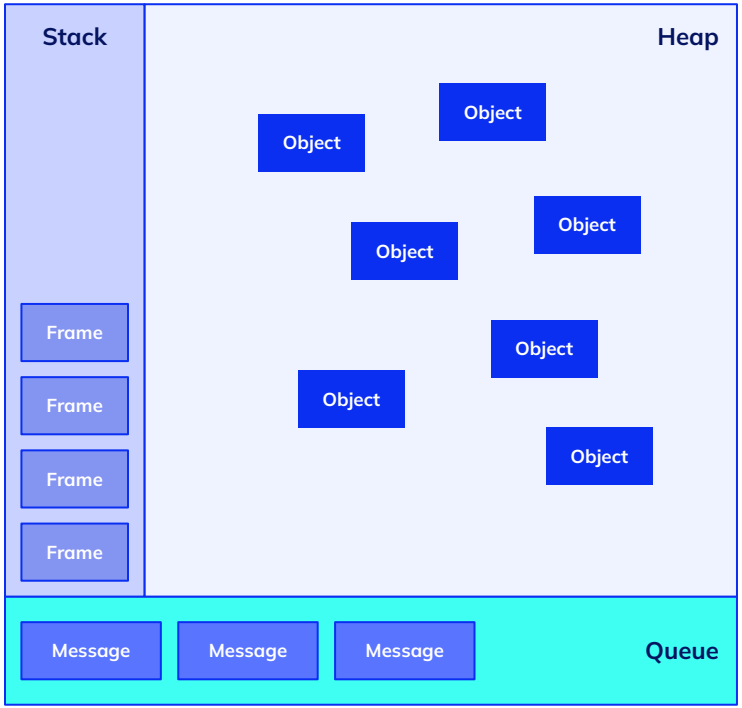
Manejo de la memoria en JavaScript

Un programa divide la memoria asignada en dos grandes partes:

- ***Heap***
  Memoria destinada para las tareas inmediatas de la carga inicial.
- ***Call-Stack***
  Memoria destinada a almacenar operaciones a ejecutarse "a destiempo" de la ejecución del script.

Básicamente, **un programa JavaScript atraviesa dos fases:**

1. Primero, **se ejecuta el script de inicio a fin, ocupa el *heap*.** En esas acciones iniciales, se pueden agregar al ***Call-Stack*** funciones para ser ejecutadas en determinadas condiciones.
2. Cuando dichas condiciones se cumplen, **pasan a la cola de llamadas *(Queue)*** para ser ejecutadas una a una.


Stack
Heap
Object
Object
Object
Object
Object
Object
Object
Frame
Frame
Frame
Frame
Message
Message
Message
Queue

Heap

La memoria dinámica que se almacena en el *heap* es aquella que se utiliza para almacenar datos que se crean en el medio de la ejecución de un programa.

En general, este tipo de datos puede llegar a ser casi la totalidad de los datos de un programa.

¿Qué se encuentra en el Heap?

- Variables y constantes.
- Funciones definidas en el ámbito global.
- Objetos creados.

Call Stack

El ***Call Stack*** (pila de llamadas) es una **estructura de datos dinámica que almacena información sobre las funciones activas** de un programa. En esta estructura, se almacena información sobre las funciones y subrutinas que se están ejecutando, que están pausadas o que deban ser ejecutadas en determinado momento.

Su principal función es la **capacidad de ejecutar varias funciones en paralelo** e ir haciendo el **seguimiento del estado** de cada una de ellas.

¿Qué se encuentra en el Call-Stack?

- **Event Listeners**.
- Funciones que siguen el patrón ***Callback***.

Proceso asincrónico

Llamamos ***proceso asincrónico*** a todo proceso que **se ejecuta desde el *call-stack*,** y no desde el heap.

Los eventos son un ejemplo. Al llamar a la función **`.addEventListener`** se registra en el **`call-stack`** una llamada pendiente a la función pasada al **`Event Listener`**.

Cuando ese evento se ejecuta, todas las funciones asociadas pasan del ***Call-Stack*** al ***Queue*** para ser ejecutadas, y usarán los recursos del ***heap***.

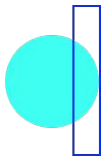
Sin embargo, el programa puede continuar sin que se ejecute dicha función. Y la llamada a esa función no bloquea la carga de la página. Es asincrónica.

Tarea asincrónica

Un proceso asincrónico es un conjunto de tareas asincrónicas.

Cada tarea representa una función a ejecutarse en el *Call-Stack*.

Un proceso sería un encadenamiento de varias tareas asincrónicas.



Por ejemplo

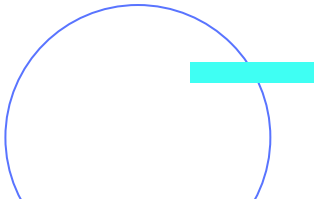
El proceso *Generar Reporte*, puede tener las siguientes tareas:

1. Obtener el listado de la API.
2. Aplicar criterios de filtrados.
3. Aplicar reducciones.
4. Convertir a JSON el resultado.

API Promise

Usamos el ***Call-Stack*** al desarrollar procesos asincrónicos, y éstos se componen de tareas asincrónicas. Ahora bien ¿cómo desarrollamos estas tareas? Mediante la API Promise.

Una promesa es una tarea asincrónica con tres estados posibles: pendiente, resuelta o rechazada.



- **Toda promesa inicia como pendiente.** Al ejecutar **`new Promise`** pasamos una función que se ejecutará automáticamente.
- Dentro de dicha función podemos definir el **estado de la promesa (resuelta / rechazada).**
- Por fuera del constructor, definimos ***callbacks*** que serán registrados en el ***call-stack*** y se ejecutarán en el cambio de estado de la promesa.

¡Sigamos trabajando!