SCOPE Y THIS

SCOPE

Scope es la accesibilidad que tienen las variables, funciones y objetos en partes específicas del código durante el tiempo de ejecución. En otras palabras, el scope determina la visibilidad de las variables en áreas del código. JavaScript cuenta con dos tipos de scope: global y local. Las variables dentro de una función se encuentran en un scope local, aquellas definidas fuera de una función están en un scope global.

SCOPE GLOBAL

Al momento de empezar a escribir en un documento de JavaScript ya estás en un scope global y solamente existe uno todo el documento. Todas las variables que estén definidas fuera de una función se encuentran en el scope global. Esto significa que se puede acceder a dichas variables e incluso modificarlas desde cualquier otra parte del código, incluso dentro de una función.

SCOPE LOCAL

Las variables definidas dentro de una función se encuentran en un scope local, esto significa que solo se puede acceder a ellas dentro de la función donde se definió la variable. Esto te permite tener variables con el mismo nombre en distintas funciones, cada una de esas variables estará ligada a su respectiva función.

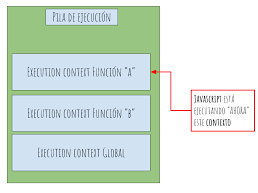
Sentencias de control de flujo y bucles como if/else, switch, for o while no crean un nuevo scope como lo hacen las funciones. Cualquier variable declarada dentro de estos bloques tendrá el mismo scope donde se declaró la sentencia.

EXECUTION CONTEXT

El ambiente en el cual cada línea de código es ejecutada se conoce como Execution Context. Cada vez que se llama o ejecuta una nueva función, JavaScript crea un nuevo execution context, estos se van apilando en lo que se conoce como Execution Stack.

Todas las variables y declaraciones de funciones por default forman parte del Global Execution Context. Al momento de ejecutar una funcion se crea un nuevo execution context encima del global, después de crear var a en este nuevo contexto se ejecuta otra funcion y el proceso se repite. El contexto que esté hasta arriba en la pila es el que se está ejecutando en el momento, una vez que la ejecución finalice se elimina de la pila y se procede a ejecutar el que sigue, así hasta terminar.

El scope se refiere a la visibilidad de variables mientras que el contexto se refiere al valor de this. Es decir, conforme cambia el execution context también lo hace el objeto this.



VALOR DE THIS

En un browser o navegador, this hace referencia a los objetos globales igual que en node.js.

THIS -> Window (Navegador)

THIS -> Global (Node.js)

THIS EN NUEVAS INSTANCIAS

Los function constructor nos permiten definir las propiedades de un objeto que podría existir eventualmente. El constructor en sí no es un objeto. Cuando usamos el keyword new dicho constructor nos retorna un objeto (instancia). En este caso this hace referencia al objeto instanciado.



CAMBIAR EL CONTEXTO DE THIS

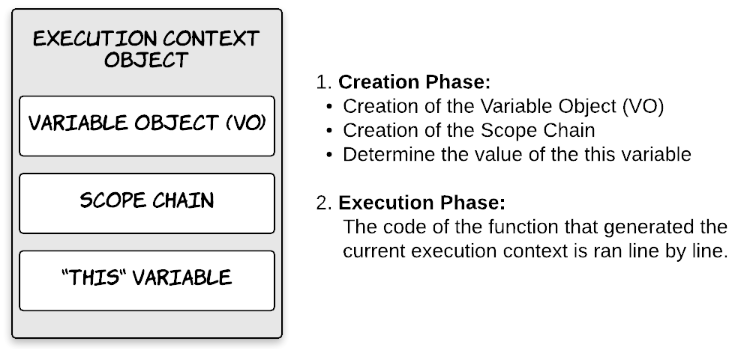
En JavaScript las funciones son un tipo de objeto. Todas las funciones tienen los métodos call, bind, y apply que nos permiten cambiar el contexto de this al momento de ejecutar la función. En la sección anterior vimos cómo this cambia para cada instancia de Person.

La única diferencia entre call y apply es la forma en que se pasan los argumentos. Con el método call pasamos los argumentos separados por coma, mientras que en el caso de apply, el segundo argumento debe ser un arreglo con los argumentos restantes.

El método bind retorna una nueva función con this haciendo referencia al primer argumento que se pasó.

EXECUTION CONTEXT OBJECT

Execution Context con un objeto que tiene tres propiedades: Variable Object, el cual contiene los argumentos de la función, declaraciones de variables internas y declaraciones de funciones. Scope Chain, este contiene el Variable Object del Execution Context actual así como el de los padres. Por último la variable this.



Cuando una función es llamada un nuevo Execution Context se coloca encima del Execution Stack y esto sucede en dos fases. La fase de creación que básicamente es crear o inicializar las propiedades descritas más arriba: Variable Object, Scope Chain y this. La segunda fase es la fase de ejecución, aquí se ejecuta el código de la función línea por línea.

VARIABLE OBJECT (VO)

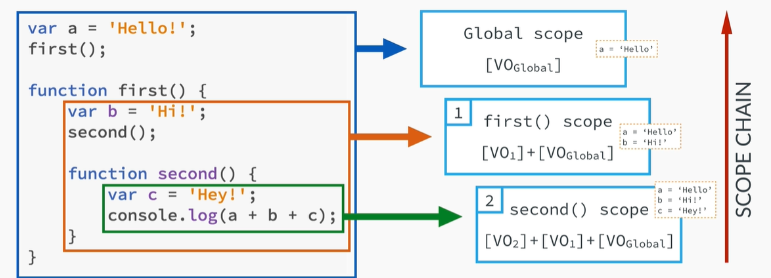
Cuando se crea el Variable Object se genera un objeto que contiene todos los argumentos que se pasaron a la función. Después, por cada declaración de función se crea una nueva propiedad dentro del VO que apunta a la función en sí. Por último, por cada declaración de variable se crea una propiedad en el VO y se inicializa la variable como undefined.

Los últimos dos pasos son lo que se conoceb como Hoisting. En este punto aún no llegamos a la fase de ejecución, es decir, el código de nuestra función aún no se ejecuta, sin embargo, ya tenemos acceso a las funciones y las variables son inicializadas con undefined.

SCOPE CHAIN

En esta fase se crea un nuevo scope por cada función. Toda función que se encuentre dentro de otra función tiene acceso al scope de la función más externa.

De esta forma la función second tiene acceso tanto al scope de la función first como el scope global. Por lo tanto tiene acceso a las variables a y b.



THIS VARIABLE

Por último se determina cuál será el valor de this. Cuando hablamos de llamadas de funciones regulares, this apunta al objeto global window. Cuando hablamos de un método entonces this apunta al objeto que está llamando el método.

Es importante tomar en cuenta que a this no se le asigna valor alguno hasta que la función que lo define es ejecutada.

DOCUMENT OBJECT MODEL

Permite renderizar el contenido de una página web y funciona a través de nodos a partir de Window

