JS_ Closures

Antonio Espín Herranz

Closures

- Una ***closure*** (cierre) es una función definida dentro del cuerpo de otra.

- Dentro de ella podemos acceder a las variables definidas dentro de la función padre.
- Un *closure* es un tipo especial de objeto que combina dos cosas: una función, y el entorno en que se creó esa función.

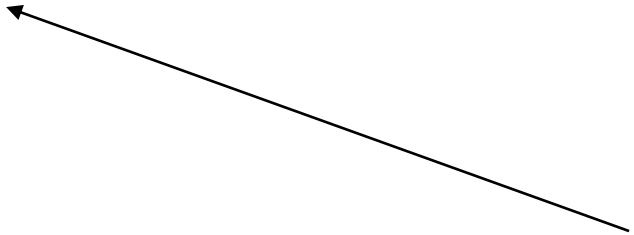
Ejemplo

- function sayHello(name) {
  var text = 'Hello ' + name;
  var sayAlert = function() { alert(text); }
  sayAlert();
  }

- La llamada sería: sayHello(‘nombre’);
- Definimos una función dentro de otra.
  - Desde la función interna accedemos a las variables de la función más externa.

Devolviendo referencias

- function sayHello2(name) {
  var text = 'Hello ' + name; // local variable
  var sayAlert = function() { alert(text); }
  return **sayAlert**;
  }



- Devuelve la referencia a la función anónima creada.
  - La podemos utilizar así: **var say2 = sayHello2('juan');**
  - Llamada a la función: **say2();**
  - Ver el código de la función:**alert(say2.toString());**

Closures

- La **closure** es una variable local para la función que la define que **permanece "viva"** incluso después de terminar la función que la define.

- **La closure se mantiene en la memoria dinámica y no se almacena en la pila.**

- Javascript almacena una referencia a la función y otra a la closure.

Ejemplo

```
function creaSumador(x) {
  return function(y) {
    return x + y;
  };
}

var suma5 = creaSumador(5);
var suma10 = creaSumador(10);

console.log(suma5(2));  // muestra 7
console.log(suma10(2)); // muestra 12
```

En este ejemplo, hemos definido una función creaSumador(x) que
toma un argumento único x y devuelve una nueva función.
Esa nueva función toma un único argumento y, devolviendo la suma de x + y.

En esencia, creaSumador es una fábrica de función:
crea funciones que pueden sumar un valor específico a su argumento.
En el ejemplo anterior utilizamos nuestra fábrica de función para
crear dos nuevas funciones: una que agrega 5 a su argumento y
otra que agrega 10.

suma5 y suma10 son ambos closures.
Comparten la misma definición
de cuerpo de función, pero almacenan diferentes entornos.
En el entorno suma5, x es 5. En lo que respecta a suma10, x es 10.

Ejemplo

- **Las variables** a las que accedemos mediante la closure **se almacenan por referencia** no se copian.

- ```
  function say123() {
    var num = 123;
    var sayAlert = function() { alert(num); }
    num++;
    return sayAlert;
  }
  ```

- ```
  var sayNum = say123(); // Capturamos la referencia.
  ```
- ```
  sayNum();                    // Muestra 124.
  ```
- ```
  alert(sayNum.toString());// Muestra el código.
  ```

Varias closures dentro de la misma función

- También podemos definir varias closures dentro de la misma función y en este caso todas las closures van a compartir las mismas variables de la función padre.

-
```
function setupSomeGlobals() {
  var num = 123;

  // Definimos variables globales con referencias a funciones
  gAlertNumber = function() { alert(num); }
  gIncreaseNumber = function() { num++; }
  gSetNumber = function(x) { num = x; }
}
```

Varias closures dentro de la misma función 2

- Podemos hacer llamadas de este estilo:

```
gAlertNumber();
gIncreaseNumber();
gSetNumber(5);
```

- Visualizar su código:

```
alert(gAlertNumber().toString());
alert(gIncreaseNumber().toString());
alert(gSetNumber(5).toString());
```

Definir funciones dentro de un bucle

```
function construyeLista(lista) {
var resultado = [];
for (var i = 0; i < lista.length;  i++) {
      var item = 'item' + lista[i];
      resultado.push( function()
      {alert(item + ' ' + lista[i])} );
} return resultado;
}

function testLista() {
var fnlista = construyeLista([1,2,3]);
for (var j = 0; j < fnlista.length;  j++)
{
fnlista[j]();
}
}
```

- Cuidado con definir funciones dentro de un bucle.
- Las 3 closures que definimos dentro acceden a la variable i **“que termina con el valor 3”.**
- El último valor “3” es con el que se quedan.

Acceso a variables después de la función

```
function decirHola() {
        var decirAlerta = function() { alert(alicia); }
            // variable local que termina estando dentro de la clausura
            var alicia = 'Hola Alicia';
            return decirAlerta;
     }

    function ejemplo6(){
            var miFuncion = decirHola();

            miFuncion();
    }
```

Aunque se define la variable alicia, después de la función anónima tenemos acceso a ella.

A tener en cuenta

- Cuando usamos function dentro de una función estamos creando una closure.

- Una clausura en javascript es como mantener copia de todas las variables locales, justo como estaban cuando la función retornó.

Pasar código a funciones

```
var eachArr = function (arr, callback) {
    for (var i = 0; i < arr.length; i++) {
            callback(arr[i]);
    }
};

// Llamamos a la función pasándole un array y la
// función a aplicar:
eachArr( [1, 2, 3], function( e ) { alert(e); } );
```

En ES6

```
var mapper2 = ((arr, callback) => {
        for (var i = 0; i < arr.length ; i++){
                callback(arr[i]);
        }
});

mapper2([1,2,3], function(e){ console.log(e);
});
```