JAVASCRIPT FUNCIONAL

Jesús Javier Doménech Arellano

27 Enero 2016

Índice

Programación Funcional

Programación Funcional

Características:

- Código inteligente.
- Menor complejidad.
- Mayor modularidad.

Herramientas:

- Mutación.
- Combinación.
- Uso de funciones.

¿Por qué Javascript?

¿Por qué Javascript (JS)?

Este lenguaje posee las siguientes características:

- Es dinamicamente tipado.
- Esta orientado a objetos (prototype).
- Funcional.
- De propósito general.

¿Por qué Javascript (JS)?

Este lenguaje posee las siguientes características:

- Es dinamicamente tipado.
- Esta orientado a objetos (prototype).
- Funcional.
- De propósito general.

Además, está presente en:

- Aplicaciones móviles.
- Sitios web.
- Servidores web.
- Aplicaciones de escritorio.
- Bases de datos.

JavaScript en 4 minutos

Expresiones:

```
1  // Números siempre son doubles
2  3; // -> 3
3  1.5; // -> 1.5
4  ((10+4*0.5-2)/2)%3; // -> 1?
5
6  // Números no reales
7  Infinity;
8  -Infinity;
9  NaN;
10
11 // Booleanos
12 true;
13 false;
14
15 // Strings
16 'hola mundo!';
17 "hola mundo!";
18
19 //Sin valor;
20 null; // no tiene valor aposta
21 undefined; // no tiene un valor en este ámbito todavía
```

Operadores:

```
1  // + - * / %
2
3  !true; // negacion
4  1 == 1; // < > <= >=
5
6  "Hola " + 'mundo!'; //contatenar
7  //tambien puedes comparar strings
8  "a" < 'b'; // -> true
9
10 //ATENCION
11 "5" == 5; // -> true NO IMPORTA EL TIPO!
12 "5" === 5; // -> false
```

Operador Ternario:

```
1  condicion ? instruccionCierto : instruccionFalso;
```

JS EN 4 MIN

Variables:

```
1  var nombreDeVariable = 2;
2
3  variableSinPoneVar = 22; // ahora es una variable global
4
5  var variableSinValor; // -> ahora vale undefined
```

Colecciones:

```
1  var miArray = [ "hola", 22, true, null, 2.2];
2
3  miArray[2]; // -> true
4
5  miArray.push("otro");
6  miArray.length; //-> 6
7
8  // diccionarios == Objetos
9  var miObjeto = {clave:"valor", "clave 2": [1,"valor"]};
10
11 miObjeto.clave; // -> "valor"
12 miObjeto["clave"]; // -> "valor"
13 miObjeto.clave3 = 4;
14 miObjeto.clave4; // -> undefined
```

Estructuras de Control:

```
1  if ( condicion ) {
2  } else if ( condicion2 ) {
3  } else {
4  }
5
6  while ( condicion ) {
7  }
8
9  do {
10 }while ( condicion )
11
12 for ( init; condicion; iteracion) {
13 }
14
15 for ( var key in variable ){  // variable puede ser un Objeto o Array
16 }
17
18 switch ( variable ) {
19   case Valor1:
20     break;
21   default:
22    break;
23 }
```

Funciones:

```
1   function (param1, param2, ...){
2      instrucción1;
3      instrucción2;
4      ...
5      return instrucciónfinal;
6   }
7
8   function Nombre( ... ){ ... }
9
10  var Nombre2 = function ( ... ) { ... };
```

¿Punto y coma?: No es necesario ponerlo siempre. Pero, ¡cuidado!

```
1   function foo (parametro){
2      return   // -> se añade un ; automaticamente
3      {
4         clave: parametro
5      }
6   }
7
8   foo(10) // -> undefined
```

JavaScript Funcional

JS Funcional

λ expresiones:

```
(param1, param2, ...) => expresión;

(param1, param2, ...) => {
                           predicado1;
                           predicado2;
                           ...
                           return predicadofinal;
                         }
```

Ejemplo:

```
    var multiplicar = (a,b) => a * b;

    var apply = (opt, a, b) => opt(a,b);
    apply(multiplicar, 4, 8); // -> 32
```

Erlang:

```
condición1 ->
               instrucciones1;
condición2 ->
               instrucciones2;
...
condiciónN ->
               instruccionesN;
true       ->
               instruccionesDefault
```

JS Funcional

Erlang:

```
condición1 ->
               instrucciones1;
condición2 ->
               instrucciones2;
...
condiciónN ->
               instruccionesN;
true       ->
               instruccionesDefault
```

Javascript:

```
 condición1 ? instrucciones1 :
 condición2 ? instrucciones2 :
 ...
 condiciónN ? instruccionesN :
              instruccionesDefault;
```

JS Funcional

"UnderScore" y "Lodash" son dos librerías que definen un entorno contenido en la variable "_". Nos proporciona diferentes métodos:

- *forEach*: recibe una lista y una función, se ejecuta la función una vez por cada elemento de la lista.
- *map*: recibe una lista y una función, devuelve el resultado de ejecutar la función a cada elemento de la lista.
- *filter*: recibe una lista y una función, devuelve los valores de la lista que dan cierto en la función pasada.
- *reduce*: recibe una lista, una función y un elemento base. La función recibe un acumulador y un elemento de la lista. Devuelve el valor final de la acumulación.
- *curry*: recibe una función con N argumentos y la currifica.

"Lazy" es una librería que nos permite realizar **evaluación perezosa**.

```
1 Lazy([1, 2, 4])        // instanceof Lazy.ArrayLikeSequence
2 Lazy({ foo: "bar" })   // instanceof Lazy.ObjectLikeSequence
3 Lazy("hello, world!") // instanceof Lazy.StringLikeSequence
4 Lazy()                 // sequence: []
5 Lazy(null)             // sequence: []
```

Nos proporciona funciones aplicables a secuencias como:

- **chunk:** agrupa una secuencia en secuencias de tamaño n.
- **compact:** quita de la secuencia *false, 0, "", null* y *undefined*.
- **reduce**, **map**, **filter**, **reverse**, **each**, **take**, ...

```
1 // Primeros 10 cuadrados divisibles por 3
2 var oneTo1000 = Lazy.range(1, 1000).toArray();
3 var sequence = Lazy(oneTo1000)
4   .map(function(x) { return x * x; })
5   .filter(function(x) { return x % 3 === 0; })
6   .take(10)
7   .each(function(x) { console.log(x); });
```

¿Más?

- *Wrappers*
- *Funtores*
- *IO Monads*
- *Acceso al DOM*

Ejemplos

forEach

```
1 function forEach(array, action) {
2   for (var i = 0; i < array.length; i++)
3     action(array[i]);
4 }
```

forEach

```
function forEach(array, action) {
  for (var i = 0; i < array.length; i++)
    action(array[i]);
}
```

Ejemplo:

```
function sum(numbers) {
  var total = 0;
  forEach(numbers, function (number) {
    total += number;
  });
  return total;
}

console.log(sum([1, 10, 100])); // -> 111
```

REDUCE

```
function reduce(action, base, array) {
  forEach(array, function (element) {
    base = action(base, element);
  });
  return base;
}
```

REDUCE

```
function reduce(action, base, array) {
  forEach(array, function (element) {
    base = action(base, element);
  });
  return base;
}
```

Ejemplo:

```
function add(a, b) {
  return a + b;
}

function sum2(numbers) {
  return reduce(add, 0, numbers);
}

console.log(sum2([1, 10, 100])); // -> 111
```

MAP

```
function map(action, array) {
  var result = [];
  forEach(array, function (element) {
    result.push(action(element));
  });
  return result;
}
```

MAP

```
1 function map(action, array) {
2   var result = [];
3   forEach(array, function (element) {
4     result.push(action(element));
5   });
6   return result;
7 }
```

Ejemplo:

```
1 console.log(map(Math.round,[0.1,9.8,Math.PI])); // -> [0, 10, 3]
```

FILTER

```
function filter(func, array) {
  var result = [];
  forEach(array, function (element) {
    if (func(element))
      result.push(element);
  });
  return result;
}
```

FILTER

```
function filter(func, array) {
  var result = [];
  forEach(array, function (element) {
    if (func(element))
      result.push(element);
  });
  return result;
}
```

Ejemplo:

```
function lt(n){
  return function(a){
    return a < n;
  }
}

console.log(filter(lt(5), [3, 9.89, 4.8])); // -> [3, 4.8]
```

COMPOSE

```
1 function compose(f, g) {
2   return function(x) {
3     return f(g(x));
4   };
5 }
```

COMPOSE

```
function compose(f, g) {
  return function(x) {
    return f(g(x));
  };
}
```

Ejemplo:

```
function f1(a){return a + ' 1';}
function f2(b){return b + ' 2';}
function f3(c){return c + ' 3';}

var composition = f3.compose(f2).compose(f1); // f3(f2(f1(___)))

console.log( composition('count') ); // -> 'count 1 2 3'
// f3(f2(f1('count'))) = f3(f2('count' + ' 1')) =
//                     = f3('count 1' + ' 2') =
//                     = 'count 1 2' + ' 3' =
//                     = 'count 1 2 3'
```

Bibliografía

Bibliografía

1. D. Mantyla, “Functional programming in JavaScript”. Packt Publishing, 2015.
2. M. Haverbeke, “Eloquent JavaScript”, A Modern Introduction to Programming
3. “Librería Lazy.js”
   `http://danieltao.com/lazy.js/`
4. “Librería Underscore.js”
   `http://underscorejs.org/`
5. Luis Atencio, “Functional Programming in Javascript”
   `https://dzone.com/storage/assets/379008-rc217-functionalprogramming.pdf`
6. “Javascript Garden”
   `http://bonsaiden.github.io/JavaScript-Garden`

¿ ?