Javascript
(API backend)

Marcelo Alaniz

FullStack Javascript Developer

Contenido

- Sintaxis y semántica Ecmascript5
- Asincronismo (callbacks, events y promises)
- Nodejs + ExpressJS
- Mejoras de Ecmascript6
- MongoDB
- Ejemplo Desarrollo Full Stack JS
- Trabajo Final

Javascript

Ecmascript5

Sintaxis y semántica

Identificadores

- Similar a C/Java:
  - Deben comenzar por una letra o por '_'
  - Pueden contener letras, dígitos y '_'
  - No pueden coincidir con las palabras reservadas

- Palabras reservadas de JavaScript

break
case, catch, continue
default, delete, do
else
finally, for, function
if, in, instanceof
new
return
switch
this, throw, try, typeof
var, void
while, with

Literales

- **Números**
  - Internamente las operaciones se realizan en punto flotante
  - Representación:
    - Enteros: `0, -1, 44, ...`
    - Decimales (*float*): `0.20, 3.1415, -3.23e+6`
    - Hexadecimal, empiezan por 0x: `0xFF, 0x1A`
- Valores lógicos (**Booleanos**)
  - true y false
- **Strings**
  - Secuencia de caracteres entre comillas dobles " o simples '
    - `"Esto es un String"`
    - `<a onclick="alert('Has pulsado el enlace')">...</a>`
  - Secuencias de escape, para representar caracteres especiales:
    - **\'** Comilla simple
    - **\"** Comilla doble
    - **\b** Retroceso
    - **\f** Salto de página
    - **\n** Salto de línea
    - **\t** Tabulación
    - **** Barra inclinada \

Variables

- JavaScript es un lenguaje débilmente tipado
  - No se especifica el tipo de las variables
  - Se deduce por el contenido de la variable y el contexto
- Para declarar una variable se usa la palabra reservada **var** seguida por una lista de nombres de variables a declarar separadas por **,**
  - El nombre de una variable puede estar formado por letras, números y los símbolos $ (dólar) y _ (guión bajo)
    - El primer carácter no puede ser un número

```
var variable1, $variable, _tmp;
var x = 1;
var y = 2;
z = x + y;   // z no está declarada pero al usarla por primera vez
             // se crea como variable global
```

- Para dejar una variable indefinida se le asigna el valor null

```
indefinida = null;
```

Expresiones

- Asignación
  - Guarda un valor específico en una variable

    ```
    var x = 0;
    ```
- Expresiones numéricas
  - Operadores aritméticos:
    - + , ++, -, --, *, /, % (módulo)
    - +=, -=, *=, /=, ^= (exponenciación), %=
- Expresiones lógicas
  - Operadores lógicos: && (and), || (or), ! (not)

Expresiones

- Expresiones de comparación
  - Operadores relacionales: ==, !=, >, <, >=, <=, ===, !==
  - Conversión automática de tipos en las comparaciones
    - JavaScript realiza conversiones automáticas entre tipos para llevar a cabo la comparación cuando sea necesario
      - Si un operando es una cadena y el otro un número, se intenta convertir la cadena a número. Si no se puede convertir la comparación devuelve false
      - Si uno de los operandos es un booleano y el otro un número se convierte el booleano a número (true 1, false 0)
    - Comparación estricta (===, !==): no se realiza conversión alguna
- Reglas de precedencia de operadores
  - () [] . (el operador punto sirve para los objetos)
  - ! - ++ --
  - * / %
  - +-
  - << >> >>> (desplazamientos a nivel de bit)
  - < <= > >=
  - == !=
  - & ^ | (lógicos a nivel de bit)
  - && || (lógicos boleanos)
  - = += -= *= /= %= <<= >>= >>>= &= ^= != (asignación)

Control de flujo

- Instrucciones condicionales
  - **if**

```
if ( condición ) {        // 0, "" y null equivalen a false
 // Instrucciones
}
else {
 // Instrucciones
}
```

  - **switch**

```
switch ( expresión ) { // La expresión devuelve un numero,
                       // un valor lógico o un string
case valor1:
          // Instrucciones caso 1
          break;  // para acabar el switch
case valor2:
          // Instrucciones caso 2
          break;
default:  // opcional
          // Instrucciones si no se diera ningún caso
}
```

Control de flujo

- Bucles
  - **for**

```
for ( inicialización; condición; actualización) {
 // Instrucciones
}
```

    - Ejemplo:

```
for (var i=0; i<array.length; i++) {
  procesa(array[i]);
}
```

  - **for in**

```
for ( indice in array ) {
 // Instrucciones
}
```

    - Ejemplo:

```
for (i in array) {
  procesa(array[i]);
}
```

Control de flujo

- Bucles
  - **while**

```
while( condición ){
// Instrucciones
}
```

    - Ejemplo:

```
while( true ) {
  bucle_infinito();
}
```

  - **do while**

```
do {
 // Instrucciones
} while( condición )
```

- Sentencias para control de bucles
  - Salir del bucle **break**
  - Saltar a la siguiente iteración **continue**

Control de flujo

- Excepciones: **try..catch**

```
try {
  // Código a ejecutar
}
catch(err) {
  // Gestión de errores
}
```

- Se puede lanzar una excepción con **throw**
  - throw excepcion

Funciones

- **function** ()

```
function nombre_funcion ( arg1, arg2, ...){
 // instrucciones
 return; // o return resultado;
}
```

  - Entre paréntesis la lista de parámetros, sin tipo, separados por comas
  - El tipo de resultado no se declara, y se devuelve con **return**

- Se pueden definir funciones anidadas

```
function hipotenusa(a, b) {
     function cuadrado(x) { return x*x; }
     return Math.sqrt(cuadrado(a) + cuadrado(b));
}
```

Funciones

- **arguments**
  - El objeto *arguments* permite acceder a los argumentos de una función como un array
    - Los argumentos se acceden con arguments[i]
    - El número de argumentos se accede con la propiedad length

```
function max( ){
   var m = Number.NEGATIVE_INFINITY;
   for(var i = 0; i < arguments.length; i++)
        if (arguments[i] > m) m = arguments[i];
   return m;
}
```

Ámbito de las variables

- Locales
  - Se definen dentro de una función con var
- Globales
  - Se definen fuera de cualquier función
    - O dentro de una función sin especificar var
- Dentro de una función una variable local prevalece sobre la global

Objetos

- Se dice que JavaScript es un lenguaje basado en objetos
  - En JavaScript no se definen clases, solo objetos
  - Es un lenguaje basado en prototipos (no basado en clases)
    - Se pueden crear objetos copiando prototipos de otros objetos

- Un objeto en JavaScript es un conjunto de variables con un nombre
  - Las variables del objeto se denominan **propiedades**
    - Las propiedades pueden ser valores de cualquier tipo de datos: arrays, funciones y otros objetos
  - Las propiedades que son funciones se llaman **métodos**

Objetos

- Se puede crear un objeto directamente indicando sus propiedades:

```
persona=new Object();
persona.nombre="Juan";
persona.id= 12893;
```

  - O en una sola instrucción, indicando las propiedades entre llaves:

```
var persona = { nombre: "Juan", id: 12893 }
```

  - O definir un **constructor**

```
function persona(nombre, id) {
 this.nombre=nombre;
 this.id=id;
}
```

    - Y así crear varios objetos:

```
var juan=new persona("Juan", 12893);
var adela=new persona("Adela", 23782);
```

- Acceso a sus propiedades:

```
nombre = persona.nombre;     // dos formas de acceder a una
nombre = persona.["nombre"]; // propiedad del objeto
```

Objetos

- Se pueden definir métodos para un objeto dentro del constructor

```
function persona(nombre, id) {
 this.nombre=nombre;
 this.id=id;

 function renombra(nombre) {
        this.nombre=nombre;
 }
}
```

- Y se invoca sobre el objeto:

```
var juan=new persona("Juan", 12893);
juan.renombra("Juanjo");
```

Tipos de objetos

- Objetos del lenguaje
  - Object, Boolean, Number, Math, Date, String, Array, RegExp
- Objetos del navegador
  - Window, Navigator, Screen, Location, History, Timing, Cookies
- Objetos DOM
  - Core DOM
    - Node, NodeList, NameNodeMap, Document, Element, Attr
  - HTML DOM
    - Document, Events, Elements
    - Anchor, Area, Base, Body, Button, Form, Frame, Frameset, Image, Input, Link, Meta, Object, Option, Select, Style, Table, Textarea
- Objetos definidos por el usuario

Objetos del lenguaje

- Dan soporte para manejar tipos básicos
- Todos los objetos del lenguaje tienen las propiedades
  - **constructor**
    - Devuelve la función que crea el objeto

      ```
      objeto.constructor
      ```

      - Devolverá algo así

        ```
        function Boolean() { [native code] }
        ```
  - **prototype**
    - Es un constructor que permite añadir propiedades y métodos al objeto
      - Se aplicará a todos los objetos de ese tipo

        ```
        Boolean.prototype.nuevaFuncion=function() {
                // código
        }
        ```
- Y los métodos
  - **toString**(): Devuelve una representación como string del objeto
  - **valueOf**(): Devuelve el valor primitivo (true/false, un número, etc.) del objeto

Objeto Boolean

- Permite convertir objetos no booleanos a booleanos
- Creación de un objeto booleano:

```
var unBooleano=new Boolean(otro);
```

- El valor será **false** si se crea con uno de los siguientes valores
  - **0**
  - **-0**
  - **null**
  - **""**
  - **false**
  - **undefined**
  - **NaN**
  - En el resto de los casos el valor será **true**

Objeto Number

- Solo hay un tipo de números que se puede escribir con o sin decimales
  - Todos los números se almacenan con 64 bits
- Creación de un objeto Number:

```
var num = new Number(valor);
```

- Propiedades:
  - **MAX_VALUE**: mayor número posible (1.7976931348623157e+308)
  - **MIN_VALUE**: menor número posible (5e-324)
  - **NEGATIVE_INFINITY**: -∞
  - **POSITIVE_INFINITY** : ∞
  - **NaN**: para indicar que el valor no es un número
- Métodos:
  - **toExponential**(x): pone el número en notación científica (1.23e+3)
  - **toFixed**(x): formatea el número con x decimales
  - **toPrecision**(x): formatea el número con longitud x

Objeto Math

- Ofrece varias operaciones matemáticas
  - Constantes matemáticas
    - **Math.E**
    - **Math.PI**
    - Math.SQRT2: raíz cuadrada de 2
    - Math.SQRT1_2: raíz cuadrada de 1/2
    - Math.LN2
    - Math.LN10
    - Math.LOG2E
    - Math.LOG10E
  - Métodos
    - **round**(decimal): redondeo
    - **random**(): devuelve un número aleatorio entre 0 y 1
    - **max**(x, y)
    - **min**(x, y)

Objetos String

- Métodos sobre strings
  - **length**: número de caracteres de un string: `s.length`
  - Concatenación de strings: operador **+**
    - Al igual que en Java, si el primer operando es un string, los demás operandos se convertirán a strings para concatenarse

      `var cad = "2"+2+2 → "222"`
  - **toUpperCase(), toLowerCase()**

    ```
    var m = "Juan";
    var m2= m.toUpperCase(); // m2 = "JUAN"
    ```
  - **charAt(posicion)**
  - **indexOf(caracter), lastIndexOf(caracter)**
    - Cuenta desde 0. Si no estuviera el carácter devuelven -1

    ```
    var posicion = m.indexOf('a'); // posicion = 2
    ```
  - **substring(inicio, final)**

    ```
    var resto = m.substring(1); // resto = "uan"
    ```
  - **split(separador)**

    ```
    var letras = m.split(""); // letras = ["J", "u", "a", "n"]
    m="Hola Juan"; palabras=m.split(" ") // palabras=["Hola","Juan"]
    ```

Objeto Date

- Proporciona la fecha y hora

```
new Date() // fecha y hora actual
new Date(milisegundos) //milisegundos desde 1 de enero 1970
new Date(string)
new Date(anno, mes, dia, horas, minutos, segundos, milisegundos)
```

- Métodos:
  - **getTime**(): devuelve el número de milisegundos desde 01.01.1970
  - getFullYear(): devuelve el año (cuatro dígitos)
  - getDate(): devuelve el día del mes (1..31)
  - getDay(): devuelve el día de la semana (0..6)
  - getHours(): devuelve la hora (0..23)
  - getMinutes(): devuelve los minutos (0..59)
    - Los equivalentes setDate, setHours, etc.
  - setFullYear(): cambia la fecha

```
d.setFullYear(2020,10,3);
```

  - toUTCString(): convierte la fecha a un string con formato de fecha de tiempo universal (`Wed, 30 Jan 2013 07:03:25 GMT`)

Arrays

- Colección de variables
  - Pueden ser todas del mismo tipo o cada una de un tipo diferente

```
var nombre_array = [valor1, valor2, ..., valorN];
var sin_inicializar = new Array(5);
```

  - Se accede a los elementos con `nombre_array[índice]`
    - índice es un valor entre 0 y N-1
- Propiedades y métodos
  - **length**: número de elementos de un array
  - **concat()**: concatenar los elementos de varios arrays

```
a1 = [1, 2, 3];
a2 = a1.concat(4, 5, 6);   // a2 = [1, 2, 3, 4, 5, 6]
a3 = a1.concat([4, 5, 6]); // a3 = [1, 2, 3, 4, 5, 6]
```

  - **pop()**: elimina y devuelve el último elemento del array
  - **push(elemento)**: añade un elemento al final del array
  - **shift()**: elimina y devuelve el primer elemento del array
  - **unshift(elemento)**: añade un elemento al principio del array
  - **reverse()**: coloca los elementos del array en el orden inverso a su posición original

```
a1.reverse();   // a1 = [3, 2, 1]
```

Objeto RegExp

- Para trabajar con expresiones regulares
  - Permite expresar patrones de caracteres y buscar correspondencias (*matching*) en un string
  - Una expresión regular consta de un patrón y modificadores:

    ```
    var er=new RegExp(patron,modificadores);
    ```
  - Se puede crear también con la notación más sencilla:

    ```
    var expresion_regular=/patron/modificadores;
    ```
- Los modificadores pueden ser
  - **i**: no diferencia mayúsculas de minúsculas
  - **g**: encuentra todas las correspondencias
- Operaciones sobre una expresión:
  - **test**(string) – devuelve true si se cumple el patrón en el string
  - **exec**(string) – devuelve el texto del texto correspondiente
    - O null si no hay ninguna correspondencia
- Ejercicio: ¿Qué devolverá el siguiente código?

```
var patron=new RegExp("e");
document.write(patron.test("JavaScript no es difícil"));
```

Objeto Global

- Hay un conjunto de propiedades y métodos que pueden accederse directamente
- Propiedades globales
  - **Infinity** – valor numérico que representa el infinito
  - **NaN** – valor que no es un número ("*Not a Number*")

    ```
    Number.NaN
    ```
  - **undefined** – una variable que no tiene asignado un valor

    ```
    var variable;
    if (variable===undefined) {
     // en este caso variable está indefinida
    }
    ```

Objeto Global

- Métodos globales
  - **eval**(string) – Evalua una cadena de texto como si fuera un programa JavaScript
  - **parseInt**(string, base) – Convierte una cadena de texto a un número entero
    - base indica el sistema de numeración (2..36) (si no se indica se puede derivar del inicio del string ("0x" hex, "0" octal, o decimal)
    - Si no puede hacer la conversión devuelve Number.NaN
  - **parseFloat**(string) – Convierte una cadena de texto a un float
  - **isNan**(valor) – Devuelve true si valor no es un número, false si lo es
  - **isFinite**(valor) – Devuelve true si su argumento no es NaN o Infinity
  - **encodeURI**(uri) – Codifica los caracteres especiales de una URI excepto , / ? : @ & = + $ #
    - Para codificar también estos se usa encodeURIComponent()
  - **decodeURI**(uri_codificada) – Descodifica una URI codificada

Ejercicio

Dado un arreglo con objetos:

- Realizar una función de busqueda sobre el arreglo dependiendo del campo
  - buscar(arreglo, { campo : valor}) => Elemento
- Realizar una función que ordene el arreglo:
  - ordenar(arreglo, campo) => arregloOrdenado
- Realizar una función que agregue un objeto nuevo:
  - agregar(arreglo, objeto) => arregloNuevo
- Realizar una función que elimine un objeto dado un _id:
  - eliminar(arreglo, _id) => arregloNuevo
- Realizar una función que una dos arreglos
  - unir(arr1, arr2) => arregloNuevo

Preguntas
Y
Respuestas