JAVASCRIPT

Autor:

Kevin Cárdenas.

2023

Índice

1. Introducción

1.1. ¿Cómo uso JavaScript?

JavaScript se puede utilizar en varios contextos, desde la creación de páginas web interactivas hasta el desarrollo de aplicaciones de servidor y de escritorio. A continuación, se detallan algunos de los métodos más comunes para utilizar JavaScript:

Incrustado en HTML

La forma más común de utilizar JavaScript es a través de la incrustación de código en HTML. En este enfoque, el código JavaScript se coloca dentro de etiquetas <script>en un archivo HTML y se ejecuta cuando se carga la página web en el navegador. Este método es útil para agregar interactividad a una página web, como validación de formularios o animaciones.

Archivo externo

Otra forma de utilizar JavaScript es a través de un archivo externo. En lugar de incrustar el código JavaScript directamente en HTML, el código se guarda en un archivo con extensión .js y se llama desde una etiqueta <script>en el archivo HTML. Este método es útil para mantener el código organizado y separado del contenido HTML.

Node.js

Node.js es una plataforma de tiempo de ejecución de JavaScript que permite la creación de aplicaciones de servidor. Con Node.js, los desarrolladores pueden utilizar JavaScript en el lado del servidor para crear aplicaciones escalables y de alta velocidad. Node.js también cuenta con un administrador de paquetes llamado npm que permite a los desarrolladores instalar y administrar fácilmente paquetes y dependencias de terceros.

Frameworks y bibliotecas

Hay varios frameworks y bibliotecas de JavaScript que permiten a los desarrolladores construir aplicaciones web complejas de manera más eficiente. Por ejemplo, React.js es un framework de JavaScript popular para construir interfaces de usuario, mientras que AngularJS es un framework de JavaScript para construir aplicaciones web dinámicas y de una sola página. También hay bibliotecas como jQuery que simplifican la manipulación del DOM y la interacción con el servidor.

Posteriormente ceremos algunoas ejemplos.

1.2. Tipos de datos

En JavaScript, existen cinco tipos de datos primitivos: **string**, **number**, **boolean**, **null** y **undefined**. Cada uno de ellos tiene su propio conjunto de propiedades y métodos que permiten realizar diferentes operaciones con ellos.

Tipo primitivo: String

El tipo **string** se utiliza para representar texto en JavaScript. Se pueden crear cadenas de texto utilizando comillas simples o dobles. Por ejemplo:

```
var nombre = "Juan";
var saludo = 'Hola, como estas?';
```

JavaScript también proporciona diferentes métodos para manipular cadenas de texto. Algunos de los métodos más comunes son `toUpperCase()` y `toLowerCase()`, que convierten una cadena de texto en mayúsculas o minúsculas respectivamente, y `concat()`, que se utiliza para concatenar dos o más cadenas de texto.

Tipo primitivo: Number

El tipo **number** se utiliza para representar números en JavaScript. Los números pueden ser enteros o decimales. Por ejemplo:

```
var edad = 25;
var precio = 12.50;
```

JavaScript proporciona diferentes operadores para realizar operaciones matemáticas con números. Algunos de los operadores más comunes son `+`, `-`, `*` y `/`, que se utilizan para realizar sumas, restas, multiplicaciones y divisiones respectivamente.

Tipo primitivo: Boolean

El tipo **boolean** se utiliza para representar valores verdaderos o falsos en JavaScript. Por ejemplo:

```
var esMayorDeEdad = true;
var esEstudiante = false;
```

Los valores booleanos son muy útiles para tomar decisiones en el código. Por ejemplo, se pueden utilizar en declaraciones `if` para realizar diferentes acciones dependiendo de si la condición es verdadera o falsa.

Tipo primitivo: Null

El tipo **null** se utiliza para representar un valor nulo en JavaScript. Se utiliza cuando una variable no tiene ningún valor asignado. Por ejemplo:

```
var miVariable = null;
```

Es importante tener en cuenta que **null** se considera un tipo de dato primitivo en JavaScript, a pesar de que técnicamente es un objeto.

Tipo primitivo: Undefined

El tipo **undefined** se utiliza para representar una variable que no ha sido definida o que no tiene ningún valor asignado. Por ejemplo:

```
var miVariable;
console.log(miVariable); // undefined
```

Es importante tener en cuenta que **undefined** es un valor, no un objeto. Sin embargo, en algunas versiones antiguas de JavaScript, se podía asignar un valor a **undefined**, lo que llevó a algunas confusiones en el código.

En JavaScript, los objetos son valores que tienen propiedades y métodos. Los objetos pueden ser predefinidos por el lenguaje (tipos objetos predefinidos) o definidos por el programador (funciones simples, clases, arrays, etc.). Además, existen objetos especiales en JavaScript, como el objeto global y el objeto prototipo.

Tipos objetos predefinidos de JavaScript

JavaScript tiene varios tipos objetos predefinidos que se utilizan comúnmente, como:

- **Object**: el objeto base en JavaScript, que tiene propiedades y métodos predefinidos.
- **Array**: un objeto que representa una lista ordenada de elementos. Los arrays en JavaScript pueden contener elementos de diferentes tipos de datos.
- **Date**: un objeto que representa una fecha y hora específicas.

- **RegExp**: un objeto que representa una expresión regular.
- **Math**: un objeto que proporciona constantes y funciones matemáticas.

A continuación, se muestran algunos ejemplos de uso de estos tipos objetos predefinidos:

```
// Object
let myObject = { name: "John", age: 30 };
console.log(myObject.name); // Output: John

// Array
let myArray = [1, "hello", true];
console.log(myArray[1]); // Output: hello

// Date
let currentDate = new Date();
console.log(currentDate.getFullYear()); // Output: 2023

// RegExp
let pattern = /hello/i;
console.log(pattern.test("Hello World")); // Output: true

// Math
console.log(Math.PI); // Output: 3.141592653589793
console.log(Math.floor(4.5)); // Output: 4
```

Tipos definidos por el programador: Funciones simples

Las funciones simples son objetos en JavaScript que tienen un nombre y un cuerpo, que se puede invocar utilizando su nombre. Las funciones simples pueden aceptar argumentos y pueden devolver valores. A continuación, se muestra un ejemplo de una función simple:

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

console.log(sum(2, 3)); // Output: 5
```

Tipos definidos por el programador: Clases

Las clases son una forma más avanzada de definir objetos en JavaScript. Las clases se utilizan para crear objetos que tienen propiedades y métodos. Una clase puede contener un constructor que se utiliza para

inicializar el objeto, así como métodos que se pueden utilizar para manipular el objeto. A continuación, se muestra un ejemplo de una clase:

```
class Person {
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }

    greet() {
        console.log(Hello, my name is ${this.name} and I'm ${this.age} years old.);
        }
    }

let john = new Person("John", 30);
john.greet(); // Output: Hello, my name is John and I'm 30 years old.
```

Tipos objeto: Arrays

Un array en JavaScript es un objeto que permite almacenar múltiples valores en una sola variable. Es una estructura de datos muy común en la programación y se utiliza para almacenar listas de elementos relacionados entre sí.

Los arrays en JavaScript son dinámicos, lo que significa que su tamaño puede cambiar durante la ejecución del programa. Además, pueden contener elementos de diferentes tipos de datos, como números, strings, booleanos y objetos.

Para crear un array en JavaScript, se utiliza la siguiente sintaxis:

```
let miArray = [valor1, valor2, valor3];
```

Los valores pueden ser de cualquier tipo de datos y se separan por comas. También es posible crear un array vacío y agregarle elementos posteriormente:

```
let miArray = [];
miArray.push(valor1);
miArray.push(valor2);
miArray.push(valor3);
```

Para acceder a los elementos de un array, se utiliza su índice, que comienza en 0. Por ejemplo:

```
let miArray = ["manzana", "pera", "banana"];
console.log(miArray[0]); // muestra "manzana"
console.log(miArray[1]); // muestra "pera"
```

```
console.log(miArray[2]); // muestra "banana"
```

También es posible modificar los valores de un array:

```
let miArray = ["manzana", "pera", "banana"];
miArray[1] = "naranja";
console.log(miArray); // muestra ["manzana", "naranja", "banana"]
```

Los arrays tienen una serie de métodos integrados que permiten realizar diversas operaciones, como agregar y eliminar elementos, buscar elementos, ordenar el array, entre otros. Algunos de estos métodos son:

- **push()**: agrega un elemento al final del array.
- **pop()**: elimina el último elemento del array y lo devuelve.
- **unshift()**: agrega un elemento al principio del array.
- **shift()**: elimina el primer elemento del array y lo devuelve.
- **splice()**: agrega o elimina elementos del array en una posición específica.
- **slice()**: devuelve una porción del array.
- **indexOf()**: devuelve el índice del primer elemento que coincide con el valor especificado.
- **sort()**: ordena los elementos del array.
- **reverse()**: invierte el orden de los elementos del array.

Tipos objetos: Tipos predefinidos de JavaScript

En JavaScript, los tipos de objeto predefinidos incluyen:

- Object
- Array
- Date
- RegExp
- Function
- Error

Tipos definidos por el programador: Funciones simples

Los tipos de objeto definidos por el programador en JavaScript incluyen las funciones simples, que se crean utilizando la sintaxis de declaración de función. Estas funciones pueden ser asignadas a variables y pasadas como argumentos a otras funciones.

Ejemplo:

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

let result = sum(2, 3);
console.log(result); // Output: 5
```

Tipos definidos por el programador: Clases

Las clases son otro tipo de objeto definido por el programador en JavaScript. Las clases permiten crear objetos que tienen propiedades y métodos, y se pueden utilizar para crear múltiples instancias de un objeto.

Ejemplo:

```
class Animal {
    constructor(name) {
        this.name = name;
    }

    speak() {
        console.log(${this.name} makes a noise.);
    }
}

class Dog extends Animal {
    constructor(name) {
        super(name);
    }

    speak() {
        console.log(${this.name} barks.);
    }
}

```

```
let dog = new Dog('Rex');
dog.speak(); // Output: "Rex barks."
```

Objetos especiales: Objeto global

El objeto global en JavaScript es el objeto que contiene todas las propiedades y métodos globales. Algunos ejemplos de propiedades y métodos globales incluyen:

- console.log()
- setTimeout()
- parseInt()

Estos métodos y propiedades se pueden utilizar en cualquier parte del código.

Objeto prototipo

El objeto prototipo es un objeto especial en JavaScript que es utilizado para proporcionar herencia a otros objetos. Cada objeto en JavaScript tiene un prototipo y hereda propiedades y métodos de su prototipo. El objeto prototipo es la base de la cadena de prototipos en JavaScript, que permite la creación de objetos con propiedades y métodos compartidos.

Cuando se crea un objeto en JavaScript, se asigna un prototipo a ese objeto. El prototipo se almacena en la propiedad interna [[Prototype]] del objeto. Cuando se intenta acceder a una propiedad o método de un objeto que no está definido en ese objeto, JavaScript buscará en la cadena de prototipos para encontrar esa propiedad o método en un objeto prototipo superior.

Por defecto, el objeto prototipo en JavaScript es el objeto Object, que tiene una serie de propiedades y métodos predefinidos que están disponibles para todos los objetos en JavaScript. Estas propiedades y métodos incluyen toString(), valueOf(), hasOwnProperty(), isPrototypeOf(), entre otros.

Es importante destacar que la modificación del objeto prototipo puede tener efectos en cadena en todos los objetos que lo heredan. Por lo tanto, se recomienda tener precaución al modificar el objeto prototipo predefinido en JavaScript.

Veamos un ejemplo de cómo se utiliza el objeto prototipo en JavaScript:

```
// Creamos un objeto persona
const persona = {
nombre: 'Juan',
edad: 30,
```

```
saludar() {
console.log(Hola, mi nombre es ${this.nombre});
}
};

// Creamos un objeto empleado que hereda del objeto persona
const empleado = Object.create(persona);
empleado.puesto = 'Desarrollador';
empleado.trabajar = function() {
console.log(${this.nombre} esta trabajando como ${this.puesto});
};

// Accedemos a propiedades y metodos de los objetos $
console.log(empleado.nombre); // Juan
empleado.saludar(); // Hola, mi nombre es Juan
empleado.trabajar(); // Juan esta trabajando como Desarrollador
```

En este ejemplo, creamos un objeto persona con propiedades nombre, edad y un método saludar(). Luego, creamos un objeto empleado que hereda del objeto persona utilizando Object.create(). Agregamos propiedades y métodos específicos de empleado al objeto empleado, como puesto y trabajar(). Finalmente, podemos acceder a las propiedades y métodos de ambos objetos utilizando la cadena de prototipos.

En resumen, el objeto prototipo es un concepto fundamental en JavaScript que permite la herencia de propiedades y métodos entre objetos. Es importante tener en cuenta que el objeto prototipo predefinido en JavaScript es el objeto Object, y que la modificación del objeto prototipo puede tener efectos en cadena en todos los objetos que lo heredan.

Objetos especiales: otros

Además de los objetos ya mencionados, JavaScript tiene otros objetos especiales que se utilizan en situaciones específicas. A continuación se presentan algunos ejemplos:

- **Objeto Date**: este objeto se utiliza para trabajar con fechas y horas. Permite crear objetos que representan una fecha y hora específicas, así como también realizar operaciones con ellas.
- **Objeto RegExp**: este objeto se utiliza para trabajar con expresiones regulares. Permite crear objetos que representan patrones de búsqueda y realizar operaciones con ellos.
- **Objeto Error**: este objeto se utiliza para manejar errores en JavaScript. Permite crear objetos que representan errores y lanzarlos en el código para indicar que algo ha salido mal.

- **Objeto Math**: este objeto se utiliza para realizar operaciones matemáticas. Contiene métodos para realizar operaciones como la raíz cuadrada, la potencia, el seno y el coseno, entre otros.
- **Objeto JSON**: este objeto se utiliza para trabajar con datos en formato JSON (JavaScript Object Notation). Permite convertir objetos JavaScript en formato JSON y viceversa.
- **Objeto Map**: este objeto se utiliza para crear mapas de clave-valor. Permite asociar valores con claves y acceder a ellos mediante las mismas.
- **Objeto Set**: este objeto se utiliza para crear conjuntos de elementos. Permite agregar elementos al conjunto y verificar si un elemento está presente en el mismo.

1.3. Funciones y uso de funciones