IES DOÑANA

2022
2023

Módulo: Desarrollo Web en Entorno Cliente

Unidad 2: Manejo de la sintaxis del lenguaje

Memoria de actividad 2

(ECMA)

Antonio Jesús Marchena Guerrero

2º DAW

02/10/2022

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL
UNIÓN EUROPEA
Fondo Social Europeo
El FSE invierte en tu futuro

Programa financiado por el Ministerio de Educación y Formación Profesional y cofinanciado por el Fondo Social Europeo

Contenido

1. Última versión del ECMA. 1

1.1. Diferencias y/o novedades respecto a la versión anterior. 1

2. Webgrafía 4

1. Última versión del ECMA.

La última versión de ECMA-262 fue aprobada el 22 de junio de 2022 que incluye las especificaciones de ECMAScript® 2023 (1, *2*, *3*).

1.1. Diferencias y/o novedades respecto a la versión anterior.

I. Nuevos miembros de las clases.

```
class MyClass {
  instancePublicField = 1;
  static staticPublicField = 2;

  #instancePrivateField = 3;
  static #staticPrivateField = 4;

  #nonStaticPrivateMethod() {}
  get #nonStaticPrivateAccessor() {}
  set #nonStaticPrivateAccessor(value) {}

  static #staticPrivateMethod() {}
  static get #staticPrivateAccessor() {}
  static set #staticPrivateAccessor(value) {}

  static {
    // Static initialization block
  }
}
```

Imagen 1: New members of

Las propiedades (ranuras públicas) ahora se pueden crear a través de:

- Campos públicos de instancia.

Campos públicos estáticos.

- Las ranuras privadas son nuevas y se pueden crear a través de:
  - Campos privados (campos privados de instancia y campos privados estáticos).
  - Métodos y accesores privados (no estáticos y estáticos).
- Bloques de inicialización estáticos.

II. Comprobaciones de ranuras privadas a través del operador in.

```
class ClassWithPrivateSlot {
  #privateSlot = true;
  static hasPrivateSlot(obj) {
    return #privateSlot in obj;
  }
}

const obj1 = new ClassWithPrivateSlot();
assert.equal(
  ClassWithPrivateSlot.hasPrivateSlot(obj1), true
);

const obj2 = {};
assert.equal(
  ClassWithPrivateSlot.hasPrivateSlot(obj2), false
);
```

Imagen 2: Private slot checks via the in

Las comprobaciones de ranuras privadas también se denominan "comprobaciones de marcas ergonómicas para campos privados". La siguiente expresión es una comprobación de este tipo – lo que determina si obj tiene una ranura privada #privateSlot.

#privateSlot in obj

Tenga en cuenta que sólo podemos referirnos a una ranura privada dentro del ámbito en el que fue declarada (*4*).

III. Uso de "await" en los niveles superiores de los módulos

```
// my-module.mjs
const response = await fetch('https://example.com');
const text = await response.text();
console.log(text);
```

Imagen 3: Top-level await in modules.

Ahora podemos utilizar await en los niveles superiores de los módulos y ya no tenemos que introducir funciones o métodos asíncronos (*6*).

IV. error.cause

```
try {
  // Do something
} catch (otherError) {
  throw new Error('Something went wrong', {cause: otherError});
}
```

Imagen 4: error.cause.

El error y sus subclases nos permiten ahora especificar qué error ha causado el actual.

La causa de un error *err* se muestra en el seguimiento de la pila y se puede acceder a ella a través de *err.cause* (*6*).

V. Método .at() de valores indexables.

```
> ['a', 'b', 'c'].at(0)
'a'
> ['a', 'b', 'c'].at(-1)
'c'
```

Imagen 5: Method .at() of indexable values.

El método .at() de los valores indexables nos permite leer un elemento en un índice determinado (como el operador de corchetes []) y admite índices negativos (a diferencia del operador de corchetes) (*6*).

Los siguientes tipos "indexables" tienen el método .at():

- String.
- Array
- Todas las clases Typed Array: Uint8Array, etc

VI. Índices de concordancia RegExp

```
const matchObj = /(a+)(b+)/d.exec('aaaabb');

assert.equal(
  matchObj[1], 'aaaa'
);
assert.deepEqual(
  matchObj.indices[1], [0, 4] // (A)
);

assert.equal(
  matchObj[2], 'bb'
);
assert.deepEqual(
  matchObj.indices[2], [4, 6] // (B)
);
```

Imagen 6: RegExp match índices.

Si añadimos la bandera (flag) /d a una expresión regular, su uso produce objetos de coincidencia que registran el índice inicial y final de cada captura de grupo (líneas A y B)

VII. Object.hasOwn(obj, propKey)

```
const proto = {
  protoProp: 'protoProp',
};
const obj = {
  __proto__: proto,
  objProp: 'objProp',
}

assert.equal('protoProp' in obj, true); // (A)

assert.equal(Object.hasOwn(obj, 'protoProp'), false); // (B)
assert.equal(Object.hasOwn(proto, 'protoProp'), true); // (C)
```

Imagen 7: Object.hasOwn(obj, propKey)

Object.hasOwn(obj, propKey) proporciona una forma segura de comprobar si un objeto obj tiene una propiedad propia (no heredada) con la clave propKey.

Observa que in detecta las propiedades heredadas (línea A), mientras que Object.hasOwn() sólo detecta las propiedades propias (líneas B y C).

2. Webgrafía

(1) https://www.ecma-international.org/publications-and-standards/standards/ecma-262/
(2) https://tc39.es/ecma262/
(3) https://2ality.com/2022/06/ecmascript-2022.html
(4) https://2ality.com/2022/06/ecmascript-2022.html#what%E2%80%99s-new-in-ecmascript-2022%3F