Objetos predefinidos en JavaScript.

Objetos Date e Intl.
Colecciones con clave: Map, Set, WeakMap, WeakSet

Jose Manuel Lopez Valero
José Luis Pérez Lara

- # Índice:
  - **Utilidad**
  - **Métodos y Propiedades**
  - **Ejemplos**

¿De que vamos ha hablar?

Objetos Date e Intl.
Colecciones con clave: Map, Set, WeakMap, WeakSet

Objeto: Date. Utilidad

La clase Date nos permite almacenar y modificar momentos fijos en el tiempo en un formato independiente.

Sobre la clase Date recae todo el trabajo con fechas en Javascript, cómo obtener una fecha, el día la hora actuales y otras cosas.

Objeto: Date. Constructor.

Constructores:

miFecha = Date(); → Contiene una cadena que representa la fecha y hora actual.

miFecha = new Date(params); → Al ser llamado sin parámetros, miFecha almacena una cadena con la fecha y hora actual.

Los parámetros que podemos darle al constructor son:

- Sin parámetros
- Valor de tiempo
- Un objeto Date
- Una cadena en formato reconocible (se puede cambiar con Date.parse())
- Componentes individuales de fecha y hora

Objeto: Date. Métodos Estáticos.

Date.now() → Devuelve el valor numérico correspondiente al actual número de milisegundos transcurridos desde el 1 de Enero de 1970, 00:00:00 UTC.

Date.parse() → Transforma la cadena que representa una fecha y retorna el número de milisegundos transcurridos desde el 1 de Enero de 1970, 00:00:00 UTC.

Date.UTC() → Acepta los mismos parámetros de la forma extendida del constructor (por ejemplo: del 2 al 7) y retorna el número de milisegundos transcurridos desde el 1 de Enero de 1970, 00:00:00 UTC.

Objeto: Date. Métodos de Instanciación.

Date.prototype.getDate() → Mes (1–31)

Date.prototype.getDay() → Día de la semana (0–6)

Date.prototype.getFullYear() → Año

Date.prototype.getHours() → Hora (0–23)

Date.prototype.getMilliseconds() → Milisegundos (0–999)

Date.prototype.getMinutes() → Minutos (0–59)

Date.prototype.getMonth() → Mes (0–11)

Date.prototype.getSeconds() → Valor numérico de la fecha especificada como el número de milisegundos transcurridos desde el 1 de Enero de 1970, 00:00:00 UTC. (Valores negativos para fechas previas.)

Objeto: Date. Métodos de Instanciación II.

Date.prototype.getTimezoneOffset() → Retorna la diferencia horaria en minutos para la hora local.

Date.prototype.getUTCDate() → Retorna el día (fecha) del mes (1–31) para la fecha especificada acorde a la hora local.

Date.prototype.getUTCDay() → Retorna el día de la semana (0–6) para la fecha especificada en hora universal.

…

Date.prototype.setDate() → Establece el día del mes para la fecha especificada acorde a la hora local.

…

Date.prototype.toDateString() → Retorna la "fecha" del objeto Date como una cadena fácil de leer por humanos 'Thu Apr 12 2018'.

Date.prototype.valueOf() → Retorna el valor primitivo de un objeto Date. Sobrescribe el método Object.prototype.valueOf().

Date.prototype.toLocaleTimeString() → Retorna una cadena con una representación sensible a la localización de la fecha basada en la configuración del sistema.

Objeto: Date. Ejemplos.

Si quieres guardar la fecha de navidad de 2019:

let navidad2019 = new Date('12-25-2019');

console.log(navidad2019); // Devuelve Wed Dec 25 2019 00:00:00 GMT+0100 (hora estándar de Europa central)

Si quieres saber qué día fue navidad en 2019:

console.log(navidad2019.getDate()); // Devuelve 25

Si quieres guardar el momento exacto en el que se ejecutó una parte del codigo:

let momentoEjecucion = new Date();

Si quieres saber cuántos milisegundos han pasado desde el 1 de enero de 1970 a las 00:00 hasta la navidad del 2019:

navidad2019.getTime();

Si quieres crear una fecha con el formato DD/MM/YYYY

let fechaFormateada = navidad2019.toLocaleDateString();

Objeto: Intl. Utilidad

El objeto de ámbito global Intl es el espacio de nombres para el API de Internacionalización de ECMAScript, éste provee comparación de cadenas y formato de números, fechas y tiempos con sensibilidad al lenguaje.

El objeto Intl nos ayuda con la localización de aplicaciones a otros idiomas y culturas.

Objeto: Intl. Propiedades y Métodos.

Propiedades:

Intl.Collator → Constructor para collators, objetos que permiten la comparación de cadenas con sensibilidad al lenguaje.

Intl.DateTimeFormat → Constructor para objetos que permiten el formato de fecha y tiempos con sensibilidad al lenguaje.

Intl.NumberFormat → Constructor para objetos que permiten el formato de números con sensibilidad al lenguaje.

Métodos:

Intl.getCanonicalLocales() → Método que retorna los nombres canónicos de las diferentes variantes de lenguaje.

Objeto: Intl. Ejemplos.

```
const number = 123456.789;

console.log(new Intl.NumberFormat('de-DE', { style:
'currency', currency: 'EUR' }).format(number));

// Salida "123.456,79 €"

— — — — — — — —

var number = 123456.789;

// En el alemán la coma se utiliza como separador
decimal y el punto para los millares

console.log(new
Intl.NumberFormat('de-DE').format(number));

// → 123.456,789
```

```
// La clave de extensión nu requiere un sistema de
numeración, p.ej. el decimal chino

console.log(new
Intl.NumberFormat('zh-Hans-CN-u-nu-hanidec').format(n
umber));

// → 一二三,四五六.七八九

— — — — — — — —

// En la mayoría de los países de lengua arábiga se
utilizan también símbolos arábigos

console.log(new
Intl.NumberFormat('ar-EG').format(number));

// → ١٢٣٤٥٦,٧٨٩
```

Colecciones con clave: Map, Set, WeakMap, WeakSet

¿Qué son?

Map y Set son dos tipos de estructuras de datos.

WeakMap y WeakSet son dos tipos de estructuras de objetos.

Map y Set

Map: al igual que object es una colección de datos identificados por claves. Pero la principal diferencia es que Map permite claves de cualquier tipo.

Set: es una colección de tipo especial: “conjunto de valores” (sin claves), donde cada valor puede aparecer solo una vez.

Utilidad

```
let map = new Map();

map.set('1', 'str1');   // un string como clave
map.set(1, 'num1');     // un número como clave
map.set(true, 'bool1'); // un booleano como
clave

// ¿recuerda el objeto regular? convertiría las
claves a string.
// Map mantiene el tipo de dato en las claves,
por lo que estas dos son diferentes:
alert( map.get(1)   ); // 'num1'
alert( map.get('1') ); // 'str1'

alert( map.size ); // 3
```

```
let set = new Set();

let john = { name: "John" };
let pete = { name: "Pete" };
let mary = { name: "Mary" };

// visitas, algunos usuarios lo hacen varias veces
set.add(john);
set.add(pete);
set.add(mary);
set.add(john);
set.add(mary);

// set solo guarda valores únicos
alert( set.size ); // 3

for (let user of set) {
  alert(user.name); // John (luego Pete y Mary)
}
```

Métodos y Propiedades

Map

new Map() – crea el mapa.

map.set(clave, valor) – almacena el valor asociado a la clave.

map.get(clave) – devuelve el valor de la clave. Será undefined si la clave no existe en map.

map.has(clave) – devuelve true si la clave existe en map, false si no existe.

map.delete(clave) – elimina el valor de la clave.

map.clear() – elimina todo de map.

map.size – tamaño, devuelve la cantidad actual de elementos.

Set

new Set(iterable) – crea el set. El argumento opcional es un objeto iterable (generalmente un array) con valores para inicializarlo.

set.add(valor) – agrega un valor, y devuelve el set en sí.

set.delete(valor) – elimina el valor, y devuelve true si el valor existía al momento de la llamada; si no, devuelve false.

set.has(valor) – devuelve true si el valor existe en el set, si no, devuelve false.

set.clear() – elimina todo el contenido del set.

set.size – es la cantidad de elementos.

WeakMap y WeakSet

La primera diferencia con Map es que las claves WeakMap deben ser objetos, no valores primitivos:

WeakSet se comporta de manera similar:

- Es análogo a Set, pero solo podemos agregar objetos a WeakSet (no primitivos).
- Existe un objeto en el conjunto mientras es accesible desde otro lugar.
- Al igual que Set, admite add, has y delete, pero no size,keys() ni iteraciones.

Utilidad

```
let john = { name: "John" };

let weakMap = new WeakMap();
weakMap.set(john, "...");

john = null; // sobreescribe la referencia

// ¡John se eliminó de la memoria!
```

```
let visitedSet = new WeakSet();

let john = { name: "John" };
let pete = { name: "Pete" };
let mary = { name: "Mary" };

visitedSet.add(john); // John nos visita
visitedSet.add(pete); // luego Pete
visitedSet.add(john); // John otra vez

// visitedSet tiene 2 usuarios ahora

// comprobar si John nos visitó?
alert(visitedSet.has(john)); // true

// comprobar si Mary nos visitó?
alert(visitedSet.has(mary)); // false

john = null;

// visitedSet se limpiará automáticamente
```

Métodos y Propiedades

weakMap

weakMap.get(clave) - devuelve el valor de la clave. Será undefined si la clave no existe en map.

weakMap.set(clave, valor) – almacena el valor asociado a la clave.

weakMap.delete(clave) – elimina el valor de la clave.

weakMap.has(clave) – devuelve true si la clave existe en map, false si no existe.

weakSet

weakSet.add(valor) - agrega un valor, y devuelve el set en sí.

weakSet.delete(clave) – elimina el valor, y devuelve true si el valor existía al momento de la llamada; si no, devuelve false.

weakSet.has(clave) – devuelve true si el valor existe en el weakset, si no, devuelve false.

Demo time

WebGrafía

MDN-Date
MDN-Intl
Lineadecodigo
LenguajeJs
Javascript.info
Jose-Aguilar

Map y Set
WeakMap y
WeakSet