> Esta página ha sido traducida del inglés por la comunidad. Aprende más y únete a la comunidad de MDN Web Docs.

Set

Baseline Widely available

El objeto `Set` le permite almacenar valores únicos de cualquier tipo, ya sea valores primitivos o referencias a objetos.

Descripción

Los objetos `Set` son colecciones de valores. Puede iterar a través de los elementos de un conjunto en orden de inserción. Un valor en un `Set` **solo puede ocurrir una vez**; es único en la colección del `Set`.

Igualdad de valores

Debido a que cada valor en el `Set` tiene que ser único, se verificará la igualdad de valores. En una versión anterior de la especificación ECMAScript, esto no se basaba en el mismo algoritmo que el utilizado en el operador `===`. Específicamente, para los `Set`, `+0` (que es estrictamente igual a `-0`) y `-0` eran valores diferentes. Sin embargo, esto se cambió en la especificación ECMAScript 2015. Consulte *"Igualdad de clave para -0 y 0"* en la tabla de compatibilidad del navegador para obtener más detalles.

`NaN` y `undefined` también se pueden almacenar en un `Set`. Todos los valores de `NaN` se equiparan (es decir, `NaN` se considera lo mismo que `NaN`, aunque `NaN !== NaN`).

Rendimiento

El método `has` de `Set` verifica si un valor está en un objeto `Set`, utilizando un enfoque que, en promedio, es más rápido que probar la mayoría de los elementos que se han agregado previamente al objeto `Set`. En particular, es, en promedio, más rápido que el método `Array.prototype.includes` cuando un objeto `Array` tiene un `length` igual al `size` de un objeto `Set`.

Constructor

`Set()` (inglés)

Crea un nuevo objeto `Set`.

Propiedades estáticas

`get Set[@@species]` (inglés)

La función del constructor que se utiliza para crear objetos derivados.

Propiedades de instancia

`Set.prototype.size`

Devuelve el número de valores en el objeto `Set`.

Métodos de instancia

`Set.prototype.add(value)`

Añade `value` al objeto `Set`. Devuelve el objeto `Set` con el valor añadido.

`Set.prototype.clear()`

Elimina todos los elementos del objeto `Set`.

`Set.prototype.delete(value)`

Elimina el elemento asociado a `value` y devuelve un booleano que afirma si un elemento se eliminó correctamente o no. `Set.prototype.has(value)` devolverá `false` después.

`Set.prototype.has(value)`

Devuelve un booleano que afirma si un elemento está presente con el valor dado en el objeto `Set` o no.

Métodos de iteración

`Set.prototype[@@iterator]()`

Devuelve un nuevo objeto iterador que genera los **values** de cada elemento del objeto `Set` en el orden de inserción.

`Set.prototype.values()`

Devuelve un nuevo objeto iterador que genera los **values** de cada elemento del objeto `Set` en el orden de inserción.

`Set.prototype.keys()`

Un alias para `Set.prototype.values()`.

`Set.prototype.entries()`

Devuelve un nuevo objeto iterador que contiene **un arreglo de** `[value, value]` para cada elemento del objeto `Set`, en orden de inserción.

Esto es similar al objeto `Map` (inglés), de modo que la *clave* de cada entrada es la misma que su *valor* para un `Set`.

`Set.prototype.forEach(callbackFn[, thisArg])` (inglés)

Llama a `callbackFn` una vez por cada valor presente en el objeto `Set`, en orden de inserción. Si se proporciona un parámetro `thisArg`, se utilizará como valor `this` para cada invocación de `callbackFn`.

Ejemplos

Utilizando el objeto Set

JS

```
const mySet1 = new Set();

mySet1.add(1); // Set [ 1 ]
mySet1.add(5); // Set [ 1, 5 ]
mySet1.add(5); // Set [ 1, 5 ]
mySet1.add("algún texto"); // Set [ 1, 5, 'algún texto' ]
const o = { a: 1, b: 2 };
mySet1.add(o);

mySet1.add({ a: 1, b: 2 }); // o está haciendo referencia a un objeto diferente,
// por lo que está bien

mySet1.has(5); // true, ya que 5 no se ha agregado al conjunto
mySet1.has(5); // true
mySet1.has(Math.sqrt(25)); // true
```

```
mySet1.has("Algún Texto".toLowerCase()); // true
mySet1.has(o); // true

mySet1.size; // 5

mySet1.delete(5); // elimina 5 del conjunto
mySet1.has(5); // false, 5 ha sido eliminado

mySet1.size; // 4, ya que acabamos de eliminar un valor

console.log(mySet1);
// imprime en consola Set(4) [ 1, "algún texto", {…}, {…} ] en Firefox
// imprime en consola Set(4) { 1, "algún texto", {…}, {…} } en Chrome
```

Iterando Set

JS

```
// iterar sobre los elementos en Set
// imprime en consola los elementos en el orden:
//   1, "algún texto", {"a": 1, "b": 2}, {"a": 1, "b": 2}
for (let item of mySet1) console.log(item)

// imprime en consola los elementos en el orden:
//   1, "algún texto", {"a": 1, "b": 2}, {"a": 1, "b": 2}
for (let item of mySet1.keys()) console.log(item)

// imprime en consola los elementos en el orden:
//   1, "algún texto", {"a": 1, "b": 2}, {"a": 1, "b": 2}
for (let item of mySet1.values()) console.log(item)

// imprime en consola los elementos en el orden:
//   1, "algún texto", {"a": 1, "b": 2}, {"a": 1, "b": 2}
// (key y value son los mismos aquí)
for (let [key, value] of mySet1.entries()) console.log(key)

// convertir el objeto Set en un objeto Array, con Array.from
const myArr = Array.from(mySet1) // [1, "algún texto", {"a": 1, "b": 2}, {"a": 1, "b": 2}]

// lo siguiente también funcionará si se ejecuta en un documento HTML
mySet1.add(document.body)
mySet1.has(document.querySelector('body')) // true

// conversión entre Set y Array
const mySet2 = new Set([1, 2, 3, 4])
mySet2.size                     // 4
[...mySet2]                     // [1, 2, 3, 4]

// la intersección se puede simular a través de
const intersection = new Set([...mySet1].filter(x => mySet2.has(x)))

// la diferencia se puede simular mediante
const difference = new Set([...mySet1].filter(x => !mySet2.has(x)))

// iterar entradas de Set con forEach()
mySet2.forEach(function(value) {
  console.log(value)
})

// 1
// 2
// 3
// 4
```

Implementación de operaciones básicas de conjuntos

JS

```
function isSuperset(set, subset) {
  for (let elem of subset) {
    if (!set.has(elem)) {
      return false;
    }
  }
  return true;
}

function union(setA, setB) {
  let _union = new Set(setA);
  for (let elem of setB) {
    _union.add(elem);
  }
  return _union;
}

function intersection(setA, setB) {
  let _intersection = new Set();
  for (let elem of setB) {
    if (setA.has(elem)) {
      _intersection.add(elem);
    }
  }
  return _intersection;
}

function symmetricDifference(setA, setB) {
  let _difference = new Set(setA);
  for (let elem of setB) {
    if (_difference.has(elem)) {
      _difference.delete(elem);
    } else {
      _difference.add(elem);
    }
  }
  return _difference;
}

function difference(setA, setB) {
  let _difference = new Set(setA);
  for (let elem of setB) {
    _difference.delete(elem);
  }
  return _difference;
}

// Ejemplos
const setA = new Set([1, 2, 3, 4]);
const setB = new Set([2, 3]);
const setC = new Set([3, 4, 5, 6]);

isSuperset(setA, setB); // devuelve true
union(setA, setC); // devuelve Set {1, 2, 3, 4, 5, 6}
intersection(setA, setC); // devuelve Set {3, 4}
symmetricDifference(setA, setC); // devuelve Set {1, 2, 5, 6}
difference(setA, setC); // devuelve Set {1, 2}
```

Relación con objetos Array

JS

```
let myArray = ["value1", "value2", "value3"];

// Use el constructor Set regular para transformar una matriz en un conjunto
let mySet = new Set(myArray);

mySet.has("value1"); // devuelve true
```

```
// Utilice el operador de dispersión para transformar un conjunto en una matriz.
console.log([...mySet]); // Le mostrará exactamente el mismo Array que myArray
```

Eliminar elementos duplicados del Array

JS

```
// Úselo para eliminar elementos duplicados del Array

const numbers = [2, 3, 4, 4, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 5, 32, 3, 4, 5];

console.log([...new Set(numbers)]);

// [2, 3, 4, 5, 6, 7, 32]
```

Relación con Strings

JS

```
let text = "India";

const mySet = new Set(text); // Set(5) {'I', 'n', 'd', 'i', 'a'}
mySet.size; // 5

// mayúsculas, minúsculas y omisión duplicada
new Set("Firefox"); // Set(7) { "F", "i", "r", "e", "f", "o", "x" }
new Set("firefox"); // Set(6) { "f", "i", "r", "e", "o", "x" }
```

Use Set para garantizar la unicidad de una lista de valores

JS

```
const array = Array.from(document.querySelectorAll("[id]")).map(function (e) {
  return e.id;
});

const set = new Set(array);
console.assert(set.size == array.length);
```

Especificaciones

Specification
ECMAScript Language Specification # sec-set-objects

Compatibilidad con navegadores

Report problems with this compatibility data on GitHub

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android
Set	Chrome 38	Edge 12	Firefox 13	Opera 25	Safari 8	Chrome 38 Android	Firefox for Android

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android
@@iterator	Chrome 43	Edge 12	Firefox 36	Opera 30	Safari 9	Chrome 43 Android	Firefox 36 for Android
@@species	Chrome 51	Edge 13	Firefox 41	Opera 38	Safari 10	Chrome 51 Android	Firefox for Android
Set() constructor	Chrome 38	Edge 12	Firefox 13	Opera 25	Safari 8	Chrome 38 Android	Firefox for Android
new Set(iterable)	Chrome 38	Edge 12	Firefox 13	Opera 25	Safari 9	Chrome 38 Android	Firefox for Android
new Set(null)	Chrome 38	Edge 12	Firefox 37	Opera 25	Safari 9	Chrome 38 Android	Firefox for Android
add	Chrome 38	Edge 12	Firefox 13	Opera 25	Safari 8	Chrome 38 Android	Firefox for Android
clear	Chrome 38	Edge 12	Firefox 19	Opera 25	Safari 8	Chrome 38 Android	Firefox for Android
delete	Chrome 38	Edge 12	Firefox 13	Opera 25	Safari 8	Chrome 38 Android	Firefox for Android
difference	Chrome 122	Edge 122	Firefox Nightly	Opera 108	Safari 17	Chrome 122 Android	Firefox for Android
entries	Chrome 38	Edge 12	Firefox 24	Opera 25	Safari 8	Chrome 38 Android	Firefox for Android
forEach	Chrome 38	Edge 12	Firefox 25	Opera 25	Safari 8	Chrome 38 Android	Firefox for Android
has	Chrome 38	Edge 12	Firefox 13	Opera 25	Safari 8	Chrome 38 Android	Firefox for Android
intersection	Chrome 122	Edge 122	Firefox Nightly	Opera 108	Safari 17	Chrome 122 Android	Firefox for Android
isDisjointFrom	Chrome 122	Edge 122	Firefox Nightly	Opera 108	Safari 17	Chrome 122 Android	Firefox for Android
isSubsetOf	Chrome 122	Edge 122	Firefox Nightly	Opera 108	Safari 17	Chrome 122 Android	Firefox for Android
isSupersetOf	Chrome 122	Edge 122	Firefox Nightly	Opera 108	Safari 17	Chrome 122 Android	Firefox for Android
Key equality for -0 and 0	Chrome 38	Edge 12	Firefox 29	Opera 25	Safari 9	Chrome 38 Android	Firefox for Android
keys	Chrome 38	Edge 12	Firefox 24	Opera 25	Safari 8	Chrome 38 Android	Firefox for Android

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android
`size`	Chrome 38	Edge 12	Firefox 19	Opera 25	Safari 8	Chrome 38 Android	Firefox 19 for Android
`symmetricDifference`	Chrome 122	Edge 122	Firefox Nightly	Opera 108	Safari 17	Chrome 122 Android	Firefox for Android
`union`	Chrome 122	Edge 122	Firefox Nightly	Opera 108	Safari 17	Chrome 122 Android	Firefox for Android
`values`	Chrome 38	Edge 12	Firefox 24	Opera 25	Safari 8	Chrome 38 Android	Firefox for Android

Tip: you can click/tap on a cell for more information.

Full support No support See implementation notes. User must explicitly enable this feature. Uses a non-standard name.

Has more compatibility info.

Véase también

- Polyfill de `Set` en `core-js`
- `Map` (inglés)
- `WeakMap`
- `WeakSet`

Help improve MDN

Was this page helpful to you?

Yes No

Learn how to contribute.

This page was last modified on 5 ago 2023 by MDN contributors.