Esta página fue traducida del inglés por la comunidad, pero no se mantiene activamente, por lo que puede estar desactualizada. Si desea ayudar a mantenerlo, descubra cómo activar las configuraciones regionales inactivas.

for...of

La sentencia **sentencia `for...of`** ejecuta un bloque de código para cada elemento de un objeto iterable, como lo son: `String`, `Array`, objetos similares a array (por ejemplo, `arguments` or `NodeList`), `TypedArray`, `Map`, `Set` e iterables definidos por el usuario.

Sintaxis

```
for (variable of iterable) {
  statement
}
```

`variable`

En cada iteración el elemento (propiedad enumerable) correspondiente es asignado a *variable*.

`iterable`

Objeto cuyas propiedades enumerables son iteradas.

Ejemplos

Iterando un `Array`

```
let iterable = [10, 20, 30];

for (let value of iterable) {
  value += 1;
  console.log(value);
}
// 11
// 21
// 31
```

Es posible usar `const` en lugar de `let` si no se va a modificar la variable dentro del bloque.

```
let iterable = [10, 20, 30];

for (const value of iterable) {
  console.log(value);
}
// 10
// 20
// 30
```

Iterando un `String`

```
let iterable = "boo";
```

```
for (let value of iterable) {
  console.log(value);
}
// "b"
// "o"
// "o"
```

Iterando un `TypedArray`

```
let iterable = new Uint8Array([0x00, 0xff]);

for (let value of iterable) {
  console.log(value);
}
// 0
// 255
```

Iterando un `Map`

```
let iterable = new Map([["a", 1], ["b", 2], ["c", 3]]);

for (let entry of iterable) {
  console.log(entry);
}
// ['a', 1]
// ['b', 2]
// ['c', 3]

for (let [key, value] of iterable) {
  console.log(value);
```

```
}
// 1
// 2
// 3
```

Iterando un `Set`

```
let iterable = new Set([1, 1, 2, 2, 3, 3]);

for (let value of iterable) {
  console.log(value);
}
// 1
// 2
// 3
```

Iterando un objeto arguments

```
(function() {
  for (let argument of arguments) {
    console.log(argument);
  }
})(1, 2, 3);

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

Iterando una colección del DOM

Iterando colecciones del DOM como un `NodeList`, el siguiente ejemplo añade la clase "read" a los

Iterando colecciones del DOM como un `NodeList`: el siguiente ejemplo añade la clase "read" a los párrafos (`<p>`) que son descendientes directos de un (`<article>`):

```
// Nota: Esto solo funcionará en plataformas que tengan
// implementado NodeList.prototype[Symbol.iterator]
let articleParagraphs = document.querySelectorAll("article > p");

for (let paragraph of articleParagraphs) {
  paragraph.classList.add("read");
}
```

Clausurando iteraciones

En los bucles `for...of`, se puede causar que la iteración termine de un modo brusco usando: `break`, `continue`[4], `throw` or `return`[5]. En estos casos la iteración se cierra.

```
function* foo(){
  yield 1;
  yield 2;
  yield 3;
};

for (let o of foo()) {
  console.log(o);
  break; // closes iterator, triggers return
}
```

Iterando generadores

También es posible iterar las nuevas funciones **generator**:

```
function* fibonacci() { // una función generador
  let [prev, curr] = [0, 1];
  while (true) {
    [prev, curr] = [curr, prev + curr];
    yield curr;
  }
}

for (let n of fibonacci()) {
  console.log(n);
  // interrumpir la secuencia en 1000
  if (n >= 1000) {
    break;
  }
}
```

No se deben reutilizar los generadores

Los generadores no deben ser reutilizados, incluso si el bucle **`for...of`** se ha terminado antes de tiempo con la sentencia break. Una vez abandonado el bucle, el generador está cerrado y tratar de iterar sobre él de nuevo no dará más resultados. Firefox no ha implementado aún este comportamiento y el generador puede ser reutilizado en contra de lo escrito en el estándar ES6 (13.7.5.13, step 5m), pero esto cambiará una vez que el bug error 1147371 haya sido corregido.

```
var gen = (function *(){
  yield 1;
  yield 2;
  yield 3;
})();
```

```
})();
for (let o of gen) {
  console.log(o);
  break;  // Finaliza la iteración
}

// El generador no debe ser reutilizado, lo siguiente no tiene sentido
for (let o of gen) {
  console.log(o); // Nunca será llamado
}
```

Iterando otros objetos iterables

Es posible, además, iterar un objeto que explicitamente implemente el protocolo iterable:

```
var iterable = {
  [Symbol.iterator]() {
    return {
      i: 0,
      next() {
        if (this.i < 3) {
          return { value: this.i++, done: false };
        }
        return { value: undefined, done: true };
      }
    };
  }
};

for (var value of iterable) {
  console.log(value);
}
```

```
// 0
// 1
// 2
```

Diferencia entre `for...of` y `for...in`

El bucle **`for...in`** `iterará sobre` **todas las propiedades de un objeto.** Más tecnicamente, iterará sobre cualquier propiedad en el objeto que haya sido internamente definida con su propiedad `[[Enumerable]]` configurada como `true`.

La sintaxis de **`for...of`** es específica para las **colecciones**, y no para todos los objetos. Esta Iterará sobre cualquiera de los elementos de una colección que tengan la propiedad `[Symbol.iterator]`.

El siguiente ejemplo muestra las diferencias entre un bucle **`for...of`** y un bucle **`for...in`**.

```
let arr = [3, 5, 7];
arr.foo = "hola";

for (let i in arr) {
   console.log(i); // logs "0", "1", "2", "foo"
}

for (let i of arr) {
   console.log(i); // logs "3", "5", "7"
}
```

Especificaciones

Especificación	Estado	Cometario

ECMAScript 2015 (6th Edition, ECMA-262) La definición de 'for...of statement' en esta especificación.	Standard	Definición inicial.
ECMAScript (ECMA-262) La definición de 'for...of statement' en esta especificación.	Living Standard	

Compatibilidad de navegadores

> **We're converting our compatibility data into a machine-readable JSON format** . This compatibility table still uses the old format, because we haven't yet converted the data it contains. **Find out how you can help! (en-US)**

- Escritorio
- Móvil

Característica	Chrome	Firefox (Gecko)	Edge	Opera	Safari
Soporte básico	38 [1] 51 [3]	13 (13) [2] [4]	12	25	7.1

Característica	Android	Chrome for Android	Firefox Mobile (Gecko)	IE Mobile	Opera Mobile	Safari Mobile

Soporte básico	5.1	38 [1]	13.0 (13) [2]	Sin soporte	?	8

[1] Desde Chrome 29 a Chrome 37 esta funcionalidad estuvo disponible al activar la opción chrome://flags/#enable-javascript-harmony: “JavaScript experimental”.

[2] Antes de Firefox 51, el uso de `for...of` usando `const` resultaba en un `SyntaxError` ("missing = in const declaration"). El problema ha sido resuelto (error 1101653).

[3] Chrome 51 añadió soporte para iterar objetos.

[4] Firefox aún permite el uso de un generador después de haber interrumpido el bucle error 1147371 . Como se vio más arriba, no se deben reutilizar los generadores.

Vea también

- `Array.prototype.forEach()`
- `Map.prototype.forEach()` (en-US)

Last modified: 7 ago 2021, by MDN contributors