

Iterator & for...of

--	--

- Iterator
- Iterator
- Iterator
- Iterator
- Iterator Generator
- return() throw()
- for...of
 -
 - Set Map
 -
 -
 -
 -
- Iterator
 - Iterator
 - Iterator

Iterator

JavaScript " "

Set

Map

Array 

Object ☐ ES6 ☐

Map Map

The diagram illustrates a memory layout. At the top, there is a label "Iterator" next to a small box containing three empty squares. Below this, there is a long horizontal row of 32 empty squares, representing a memory array. At the bottom, the word "Iterator" is written again, followed by a box containing 16 empty squares, which is aligned with the first half of the long array above.

Iterator

ES6

for...of

Iterator

for...of

next

next

1

2 next

3 next

4 next

next

value done

value done

next

```
var it = makeIterator(['a', 'b']);

it.next() // { value: "a", done: false }
it.next() // { value: "b", done: false }
it.next() // { value: undefined, done: true }
```

```
function makeIterator(array) {
  var nextIndex = 0;
  return {
    next: function() {
      return nextIndex < array.length ?
        {value: array[nextIndex++], done: false} :
        {value: undefined, done: true};
    }
  };
}
```

makeIterator

['a', 'b']

it

next

next

a

b

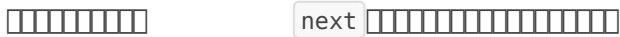
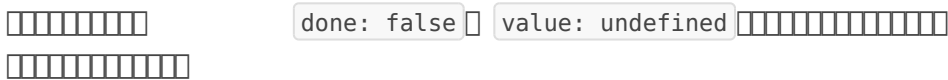
next

value done

value

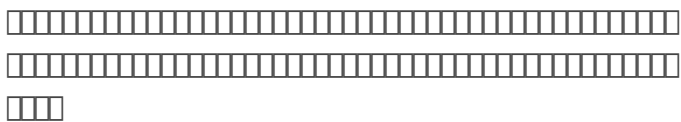
done

next



```
function makeIterator(array) {
  var nextIndex = 0;
  return {
    next: function() {
      return nextIndex < array.length ?
        {value: array[nextIndex++]} :
        {done: true};
    }
  };
}
```

Iterator



```
var it = idMaker();

it.next().value // 0
it.next().value // 1
it.next().value // 2
// ...

function idMaker() {
  var index = 0;

  return {
    next: function() {
      return {value: index++, done: false};
    }
  };
}
```



TypeScript

Iterable

Iterator

next

```
interface Iterable {
  [Symbol.iterator]() : Iterator,
}

interface Iterator {
  next(value?: any) : IterationResult,
}

interface IterationResult {
  value: any,
  done: boolean,
}
```

Iterator

Iterator

for...of

for...of

Iterator

Iterator

"

" iterable

ES6

Iterator

Symbol.iterator

Symbol.iterator

"

" iterable

Symbol.iterator

Symbol.iterator

Symbol

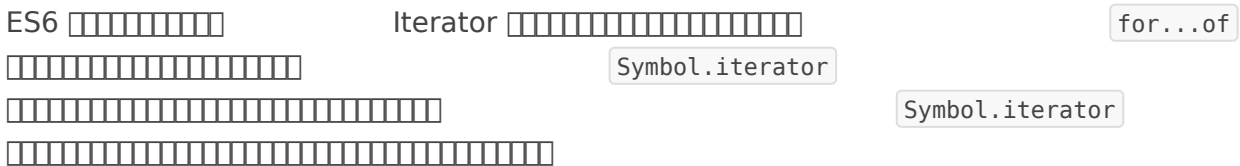
iterator

Symbol

Symbol

```
const obj = {
  [Symbol.iterator]: function () {
    return {
      next: function () {
        return {
          value: 1,
          done: true
        };
      }
    };
  };
}
```

```
};
```



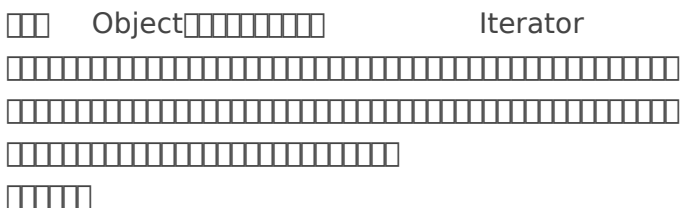
Iterator

- Array
- Map
- Set
- String
- TypedArray
- arguments
- NodeList

Symbol.iterator

```
let arr = ['a', 'b', 'c'];
let iter = arr[Symbol.iterator]();

iter.next() // { value: 'a', done: false }
iter.next() // { value: 'b', done: false }
iter.next() // { value: 'c', done: false }
iter.next() // { value: undefined, done: true }
```



Map ES5 Map ES6


```

var current = this;

function next() {
  if (current) {
    var value = current.value;
    current = current.next;
    return { done: false, value: value };
  }
  return { done: true };
}

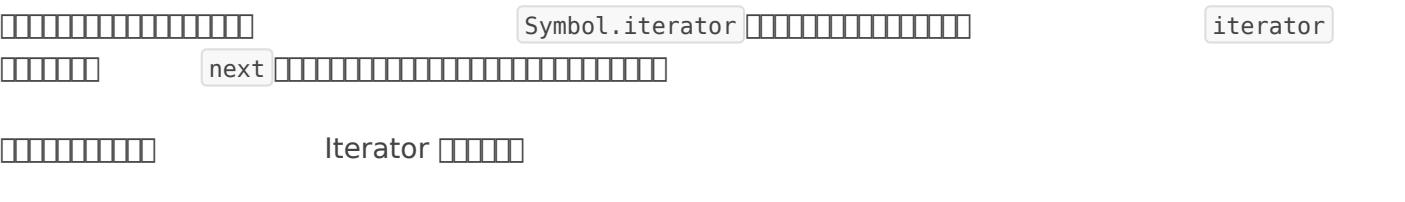
return iterator;
}

var one = new Obj(1);
var two = new Obj(2);
var three = new Obj(3);

one.next = two;
two.next = three;

for (var i of one){
  console.log(i); // 1, 2, 3
}

```



```
let obj = {
  data: [ 'hello', 'world' ],
  [Symbol.iterator]() {
    const self = this;
    let index = 0;
    return {
      next() {
        if (index < self.data.length) {
          return {
            value: self.data[index++],
            done: false
          }
        }
      }
    }
  }
}
```



```
[Symbol.iterator]: Array.prototype[Symbol.iterator]
};
for (let item of iterable) {
  console.log(item); // undefined, undefined, undefined
}
```

Symbol.iterator

```
var obj = {};

obj[Symbol.iterator] = () => 1;

[...obj] // TypeError: [] is not a function
```

obj Symbol.iterator

for...of while

```
var $iterator = ITERABLE[Symbol.iterator]();
var $result = $iterator.next();
while (!$result.done) {
  var x = $result.value;
  // ...
  $result = $iterator.next();
}
```

ITERABLE \$iterator
next done
next

Iterator

Iterator Symbol.iterator for...of

1

Set Symbol.iterator

```
let set = new Set().add('a').add('b').add('c');
```

```
let [x,y] = set;
// x='a'; y='b'

let [first, ...rest] = set;
// first='a'; rest=['b','c'];
```

2

... Iterator

```
// 
var str = 'hello';
[...str] // ['h','e','l','l','o']

// 
let arr = ['b', 'c'];
['a', ...arr, 'd']
// ['a', 'b', 'c', 'd']
```

Iterator

Iterator
Iterator

```
let arr = [...iterable];
```

3 yield*

yield*

```
let generator = function* () {
  yield 1;
  yield* [2,3,4];
  yield 5;
};

var iterator = generator();

iterator.next() // { value: 1, done: false }
iterator.next() // { value: 2, done: false }
iterator.next() // { value: 3, done: false }
```

```

iterator.next() // { value: 4, done: false }
iterator.next() // { value: 5, done: false }
iterator.next() // { value: undefined, done: true }

```

4

- for...of
- Array.from()
- Map(), Set(), WeakMap(), WeakSet()
- Promise.all()
- Promise.race()

new Map([['a',1],['b',2]])

Iterator

Iterator

```

var someString = "hi";
typeof someString[Symbol.iterator]
// "function"

var iterator = someString[Symbol.iterator]();

iterator.next() // { value: "h", done: false }
iterator.next() // { value: "i", done: false }
iterator.next() // { value: undefined, done: true }

```

Symbol.iterator

next

Symbol.iterator

```

var str = new String("hi");

[...str] // ["h", "i"]

str[Symbol.iterator] = function() {
  return {
    next: function() {

```

```

    if (this._first) {
      this._first = false;
      return { value: "bye", done: false };
    } else {
      return { done: true };
    }
  },
  _first: true
};

[...str] // ["bye"]
str // "hi"

```

██████████
str
Symbol.iterator
████████████████████
...
██████████
bye

██████████
hi

Iterator Generator

Symbol.iterator()
████████████████████
Generator

```

let myIterable = {
  [Symbol.iterator]: function* () {
    yield 1;
    yield 2;
    yield 3;
  }
};

[...myIterable] // [1, 2, 3]

// 1 2 3

let obj = {
  * [Symbol.iterator]() {
    yield 'hello';
    yield 'world';
  }
};

```

```
for (let x of obj) {
  console.log(x);
}
// "hello"
// "world"
```

□□□□□

Symbol.iterator() □□□□□□□□□□

yield □□□□□□□□□□

□□□□□□□

return() □ throw()

□□□□□□□

next() □□□□□□□

return() □□□

throw()

□□□□□□□□□□□□□□□□

next() □□□□□□□□

return() □□□

throw()

□□□□□□□□

return() □□□□□□□□□□

for...of □□□□□□□□□□□□□□□□

break

□□□□□□□

return() □□□□□□□□□□□□□□□□□□□□□□□□□□□□

return()

□□□

```
function readLinesSync(file) {
  return {
    [Symbol.iterator]() {
      return {
        next() {
          return { done: false };
        },
        return() {
          file.close();
          return { done: true };
        }
      };
    },
  };
};
```

□□□□□□□

readLinesSync □□□□□□□□□□□□□□□□□□□□□□□□□□□□

next()

□□□□□□□

return() □□□□□□□□□□□□□□□□

return() □□□

```
// □□□
for (let line of readLinesSync(fileName)) {
  console.log(line);
}
```

```

    break;
}

// 断言
for (let line of readLinesSync(fileName)) {
    console.log(line);
    throw new Error();
}

```

```

return()

```

```

return()

```

```

return()

```

Generator

```

throw()

```

Generator

Generator

for...of

ES6 C++ Java C# Python

```

for...of

```

```


```

```

Symbol.iterator

```

```

iterator

```

```

for...of

```

```


```

```

for...of

```

```

Symbol.iterator

```

```

for...of

```

Set Map

```

arguments

```

DOM NodeList

Generator

```


```

```

iterator

```

```

Symbol.iterator

```

```

for...of

```

```

const arr = ['red', 'green', 'blue'];

```

```

for(let v of arr) {
    console.log(v); // red green blue
}

```

```

const obj = {};
obj[Symbol.iterator] = arr[Symbol.iterator].bind(arr);

```

obj arr Symbol.iterator obj for...of arr

```
let arr = [3, 5, 7];  
arr.foo = 'hello';  
  
for (let i in arr) {  
    console.log(i); // "0", "1", "2", "foo"  
}
```

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

`for...of` `arr` `foo`

Set ☐ Map ☐

Set ☐ Map ☐ Iterator ☐ for...of ☐

```
var engines = new Set(["Gecko", "Trident", "Webkit", "Webkit"]);
for (var e of engines) {
  console.log(e);
}
// Gecko
// Trident
// Webkit

var es6 = new Map();
es6.set("edition", 6);
es6.set("committee", "TC39");
es6.set("standard", "ECMA-262");
for (var [name, value] of es6) {
  console.log(name + ": " + value);
}
// edition: 6
// committee: TC39
// standard: ECMA-262
```


 Set
 Map

_____ Set


```
let map = new Map().set('a', 1).set('b', 2);
for (let pair of map) {
  console.log(pair);
}
// ['a', 1]
```

```
// ['b', 2]

for (let [key, value] of map) {
  console.log(key + ' : ' + value);
}
// a : 1
// b : 2
```



ES6 `Set` `Map`

- `entries()` `[[], []]`

`entries` `[]`
- `keys()` `[[], []]`
- `values()` `[[], []]`

`Set` `Map` `Iterator`



```
let arr = ['a', 'b', 'c'];
for (let pair of arr.entries()) {
  console.log(pair);
}
// [0, 'a']
// [1, 'b']
// [2, 'c']
```



`for...of` `[[], []]`

DOM `NodeList` `[]`

`arguments`

```
// []
let str = "hello";

for (let s of str) {
  console.log(s); // h e l l o
}
```

```
// DOM NodeList[]
let paras = document.querySelectorAll("p");

for (let p of paras) {
  p.classList.add("test");
}

// arguments[]
function printArgs() {
  for (let x of arguments) {
    console.log(x);
  }
}
printArgs('a', 'b');
// 'a'
// 'b'
```

████████

for...of ████████████████████

32 █ UTF-16 ███

```
for (let x of 'a\uD83D\uDC0A') {
  console.log(x);
}
// 'a'
// '\uD83D\uDC0A'
```

██████████████████

Iterator ████████████████████

Array.from

████████

```
let arrayLike = { length: 2, 0: 'a', 1: 'b' };

// []
for (let x of arrayLike) {
  console.log(x);
}

// []
for (let x of Array.from(arrayLike)) {
  console.log(x);
}
```



□□□□□□

for...of

□□□□□□□□□□□□

Iterator

□□□□□□□□□□□□

for...in

□□□□□□□□

```
let es6 = {
  edition: 6,
  committee: "TC39",
  standard: "ECMA-262"
};

for (let e in es6) {
  console.log(e);
}
// edition
// committee
// standard

for (let e of es6) {
  console.log(e);
}
// TypeError: es6[Symbol.iterator] is not a function
```

□□□□□□□□□□

for...in

□□□□□□

for...of

□□□□

□□□□□□

Object.keys

□□□□□□□□□□□□□□□□

```
for (var key of Object.keys(someObject)) {
  console.log(key + ': ' + someObject[key]);
}
```

□□□□□□

Generator

□□□□□□□□

```
const obj = { a: 1, b: 2, c: 3 }

function* entries(obj) {
  for (let key of Object.keys(obj)) {
    yield [key, obj[key]];
  }
}
```

```
for (let [key, value] of entries(obj)) {  
  console.log(key, '->', value);  
}  
  
// a -> 1  
// b -> 2  
// c -> 3
```



JavaScript for

```
for (var index = 0; index < myArray.length; index++) {  
    console.log(myArray[index]);  
}
```

 `forEach` 

```
myArray.forEach(function (value) {  
    console.log(value);  
});
```

[illegible]

```
for...in
```

```
for (var index in myArray) {  
    console.log(myArray[index]);  
}
```

for...in 

- `for...in` `["0", "1", "2"]`
- `for...in` `["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]`
- `for...in` `["0", "1", "2"]`

for...in

`for...of`

```
for (let value of myArray) {  
  console.log(value);  
}
```

```
}
```

- `for...in` `for...in`
- `forEach` `break` `continue` `return`
-

`break` `for...of`

```
for (var n of fibonacci) {  
  if (n > 1000)  
    break;  
  console.log(n);  
}
```

`for...of` `break` `for...of` `break`

Iterator

ES2015 Iterator

```
class MyIterator extends Iterator {  
  next() {  
    // ...  
  }  
}
```

`next()`

Iterator

Iterator

- `Iterator.from()`
- `Iterator.concat()`

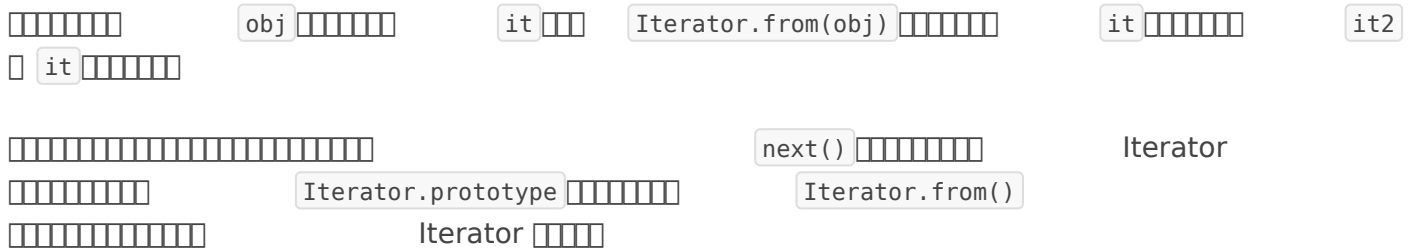
1 `Iterator.from()`

```
const it = (function* () {
  yield 1;
  yield 2;
  yield 3;
})();

const obj = {
  [Symbol.iterator]() {
    return it;
  },
};

const it2 = Iterator.from(obj);

it2 === it // true
```



```
const it = {
  current: 0,
  next() {
    return { value: this.current++, done: false };
  },
};

const obj = {
  [Symbol.iterator]() {
    return it;
  },
};

const it2 = Iterator.from(obj);

it2 === it // false
it2.next() // {value: 0, done: false}
it.next() // {value: 1, done: false}
```

```

obj
  it
    next()
      Iterator
        it
          it2
            it2.next()
              it.next()

```

```

Iterator.from()
[System.iterator]()
  Iterator

```

```

Iterator.from()

```

```

// 测试
const obj1 = (function* () {
  yield 1;
  yield 2;
  yield 3;
})();

const obj2 = Iterator.from(obj1);
obj2 === obj1 // true

// 测试
const obj3 = {
  current: 0,
  next() {
    return { value: this.current++, done: false };
  },
};

const obj4 = Iterator.from(obj4);
obj3 === obj4 // false

```

```

Obj1
obj3
  Iterator.from()
Obj1
  Iterator
  Obj3
    Iterator

```

2. Iterator.concat()

```

Iterator.concat()

```

```

const a1 = [1,2];
const a2 = [3,4];
const a3 = [5,6];

```

```
const it = Iterator.concat(a1, a2, a3)
```

```
it.next() // {value: 1, done: false}
```

```
it.next() // {value: 2, done: false}
```

```
it.next() // {value: 3, done: false}
```

|||||

Iterator.concat() | a1 | a2 | a3 |||||

|||||

```
const array = [1, 2, 3];
```

```
const set = new Set([4, 5, 6]);
```

```
function* gen() {
```

```
  yield 7;
```

```
  yield 8;
```

```
  yield 9;
```

```
}
```

```
const it = Iterator.concat(array, set, gen());
```

```
[...it] // [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

|||||

Iterator.concat() |||||

|||||

```
const obj = {
```

```
  next() {
```

```
    return { done: true };
```

```
  },
```

```
};
```

```
const it = Iterator.concat(obj); // ||
```

|||||

obj |||||

Iterator.concat() |||||

Iterator.from()

|||||

```
const obj = {
```

```
  next() {
```

```
    return { done: true };
```

```
  },
```

```
};
```

```
const it = Iterator.concat(Iterator.from(Obj)); // []
```

Iterator []

Iterator

[[[]]]

- Iterator.prototype.every()
- Iterator.prototype.filter()
- Iterator.prototype.find()
- Iterator.prototype.flatMap()
- Iterator.prototype.forEach()
- Iterator.prototype.includes()
- Iterator.prototype.join()
- Iterator.prototype.map()
- Iterator.prototype.reduce()
- Iterator.prototype.some()

[[[]]]

```
const result = Iterator.from(array)
  .map(x => x * 2)
  .filter(x => x > 10)
  .take(3)
  .toArray();
```

[[[]]]

```
const arr = ['a', '', 'b', '', 'c', '', 'd', '', 'e'];

arr.values() // creates an iterator
  .filter(x => x.length > 0)
  .drop(1)
  .take(3)
  .map(x => `${x}=`)
  .toArray()
// ['b=', 'c=', 'd=']
```

arr values() for...of

 Iterator

- `Iterator.prototype.drop(limit)`  limit 
- `Iterator.prototype.take(limit)`  limit 
- `Iterator.prototype.toArray()` 

□□□ #2

□ □□ □□ 2020-04-16 10:29:52 UTC

□ □□ □□ 2026-09-24 17:10:01 UTC