

Generator



- Array
- Array
 - Array
 - Array
 - Promise
- Generator Array
 - Array
 - Array Generator Array
 - Generator Array
 - Array
- Thunk Array
 - Array
 - Thunk Array
 - JavaScript Array Thunk Array
 - Thunkify Array
 - Generator Array
 - Thunk Array
- co Array
 - Array
 - co Array
 - Array Promise Array
 - co Array
 - Array
 - Array Stream

JavaScript

[illegible]

“”

Generator 



ES6

- 
- 
-  / 
- Promise 

Generator  JavaScript 

[illegible]

A number line starting at 0 and ending at 100. Major tick marks are labeled every 10 units (0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100). There are also minor tick marks between the major ones, representing 1-unit intervals. A blue arrow starts at 0 and points to 10, indicating a jump of 10 units.



JavaScript



The diagram illustrates the memory layout for JavaScript. It consists of two horizontal rows of small squares, each representing a 32-bit block. The first row contains 16 blocks, and the second row contains 16 blocks, totaling 32 blocks. The label 'JavaScript' is positioned above the first row.

callback

--	--	--	--	--	--

 "

--	--	--	--

 "

--

```
fs.readFile('/etc/passwd', 'utf-8', function (err, data) {
  if (err) throw err;
  console.log(data);
});
```

readFile /etc/passwd

Node null err

Promise

A B

```
fs.readFile(fileA, 'utf-8', function (err, data) {
  fs.readFile(fileB, 'utf-8', function (err, data) {
    // ...
  });
});
```

" callback hell"

Promise

Promise

```
var readFile = require('fs-readfile-promise');

readFile(fileA)
  .then(function (data) {
    console.log(data.toString());
  })
  .then(function () {
    return readFile(fileB);
  })
  .then(function (data) {
    console.log(data.toString());
  })
  .catch(function (err) {
    console.log(err);
  });
```

```
});
```

Promise

fs-readfile-promise

Promise

Promise

readFile

then

Promise

catch

Promise

Promise

Promise

then

Promise

Promise

then

Promise

Promise

Generator

yield

yield

yield

"yield" is a coroutine

yield

• yield A

• yield A

B

• yield B

• yield A

yield A

yield

```
function* asyncJob() {  
  // ...  
  var f = yield readFile(fileA);  
  // ...  
}
```

yield

asyncJob

yield

yield

yield

yield

yield

yield

yield

yield

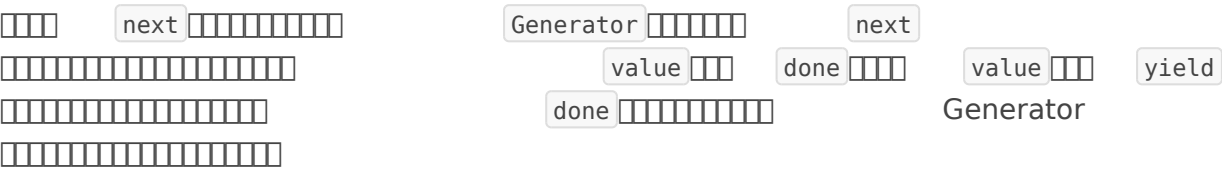
Generator

Generator ES6

Generator yield Generator

```
function* gen(x) {
  var y = yield x + 2;
  return y;
}

var g = gen(1);
g.next() // { value: 3, done: false }
g.next() // { value: undefined, done: true }
```



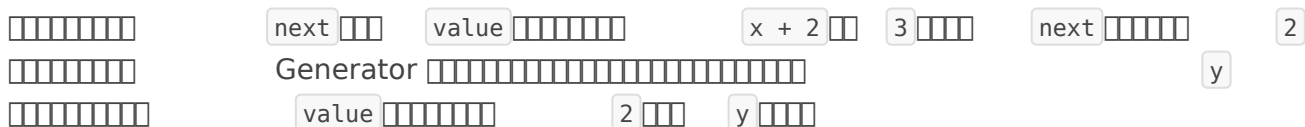
Generator

Generator

next value Generator next Generator

```
function* gen(x){
  var y = yield x + 2;
  return y;
}
```

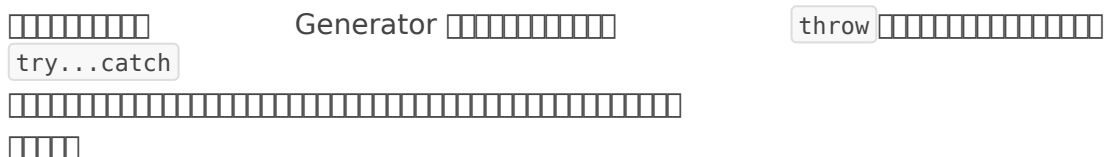
```
var g = gen(1);
g.next() // { value: 3, done: false }
g.next(2) // { value: 2, done: true }
```



Generator

```
function* gen(x){
  try {
    var y = yield x + 2;
  } catch (e){
    console.log(e);
  }
  return y;
}
```

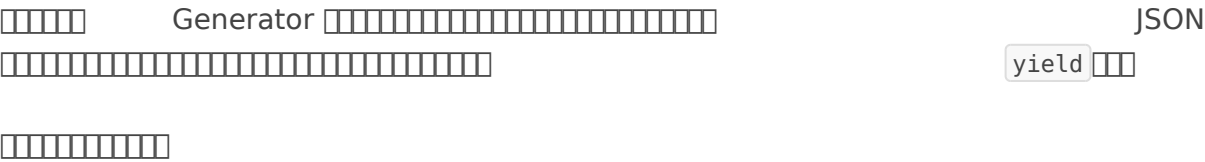
```
var g = gen(1);
g.next();
g.throw('');
//
```



Generator

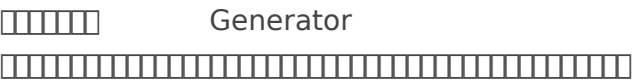
```
var fetch = require('node-fetch');

function* gen(){
  var url = 'https://api.github.com/users/github';
  var result = yield fetch(url);
  console.log(result.bio);
}
```



```
var g = gen();
var result = g.next();

result.value.then(function(data){
  return data.json();
}).then(function(data){
  g.next(data);
});
```



Thunk

Thunk Generator



Thunk 60



```
var x = 1;

function f(m) {
  return m * 2;
}

f(x + 5)
```



```
// 测试

var thunk = function () {
    return x + 5;
};

function f(thunk) {
    return thunk() * 2;
}
```

测试
f
x + 5
Thunk

测试
Thunk
"
"

JavaScript Thunk

JavaScript
 Thunk
 JavaScript
 Thunk

```
// 测试readFile
fs.readFile(fileName, callback);

// Thunk测试readFile
var Thunk = function (fileName) {
    return function (callback) {
        return fs.readFile(fileName, callback);
    };
};

var readFileThunk = Thunk(fileName);
readFileThunk(callback);
```

测试
fs
readFile

Thunk

Thunk
Thunk

```
// ES5
var Thunk = function(fn){
```

```

return function (){
  var args = Array.prototype.slice.call(arguments);
  return function (callback){
    args.push(callback);
    return fn.apply(this, args);
  }
};

// ES6
const Thunk = function(fn) {
  return function (...args) {
    return function (callback) {
      return fn.call(this, ...args, callback);
    }
  };
};

```

■■■■■■■■■■

`fs.readFile` ■ Thunk ■■■

```

var readFileThunk = Thunk(fs.readFile);
readFileThunk(fileA)(callback);

```

■■■■■■■■■■

```

function f(a, cb) {
  cb(a);
}
const ft = Thunk(f);

ft(1)(console.log) // 1

```

Thunkify ■■

■■■■■■■■■■

Thunkify ■■■

■■■■■

```
$ npm install thunkify
```

■■■■■

```
var thunkify = require('thunkify');
var fs = require('fs');

var read = thunkify(fs.readFile);
read('package.json')(function(err, str){
  // ...
});
```

Thunkify ■■■■■■■■■■■■■■■■■■■■

```
function thunkify(fn) {
  return function() {
    var args = new Array(arguments.length);
    var ctx = this;

    for (var i = 0; i < args.length; ++i) {
      args[i] = arguments[i];
    }

    return function (done) {
      var called;

      args.push(function () {
        if (called) return;
        called = true;
        done.apply(null, arguments);
      });

      try {
        fn.apply(ctx, args);
      } catch (err) {
        done(err);
      }
    }
  };
}
```

■■■■■■■■■■■■■■■■■■■■
■■■■■■■■■■■■■■■■■■■■

called ■■■■■■■■■■■■■■■■■■■■

Generator

```
function f(a, b, callback){
  var sum = a + b;
  callback(sum);
  callback(sum);
}

var ft = thunkify(f);
var print = console.log.bind(console);
ft(1, 2)(print);
// 3
```

thunkify

Generator

Think 

Generator 

ES6 ☐ Generator ☐ Thunk

Generator

--	--	--	--	--	--	--	--

```
function* gen() {  
    // ...  
}  
  
var g = gen();  
var res = g.next();  
  
while(!res.done){  
    console.log(res.value);  
    res = g.next();  
}
```

Generator ☐ gen ☒

Thunk Generator

```
var fs = require('fs');  
var thunkify = require('thunkify');  
var readFileThunk = thunkify(fs.readFile);
```

```
var gen = function* (){
  var r1 = yield readFileThunk('/etc/fstab');
  console.log(r1.toString());
  var r2 = yield readFileThunk('/etc/shells');
  console.log(r2.toString());
};
```

⬜⬜⬜⬜ yield ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜
Generator ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜
Generator ⬜⬜

⬜⬜⬜⬜ Thunk ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜
Generator
⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜
Generator ⬜⬜

```
var g = gen();

var r1 = g.next();
r1.value(function (err, data) {
  if (err) throw err;
  var r2 = g.next(data);
  r2.value(function (err, data) {
    if (err) throw err;
    g.next(data);
  });
});
```

⬜⬜⬜⬜⬜⬜ g Generator ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜
next
⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜
value ⬜⬜
done ⬜⬜⬜

⬜⬜⬜⬜⬜⬜⬜⬜⬜
Generator ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜
next
⬜⬜
value ⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜⬜

Thunk ⬜⬜⬜⬜⬜⬜⬜⬜

Thunk ⬜⬜⬜⬜⬜⬜⬜⬜⬜
Generator ⬜⬜⬜⬜⬜⬜⬜⬜
Thunk ⬜⬜
Generator

```
function run(fn) {
  var gen = fn();

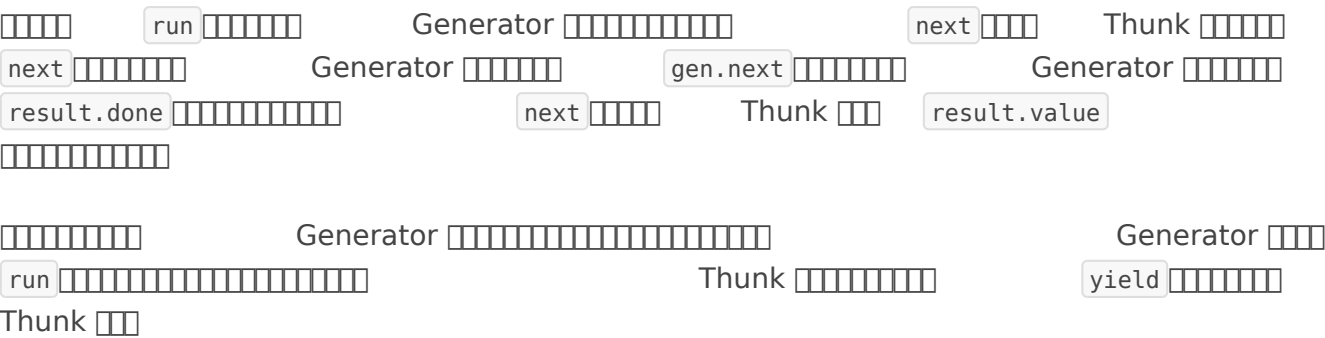
  function next(err, data) {
    var result = gen.next(data);
```

```
    if (result.done) return;
    result.value(next);
  }

  next();
}

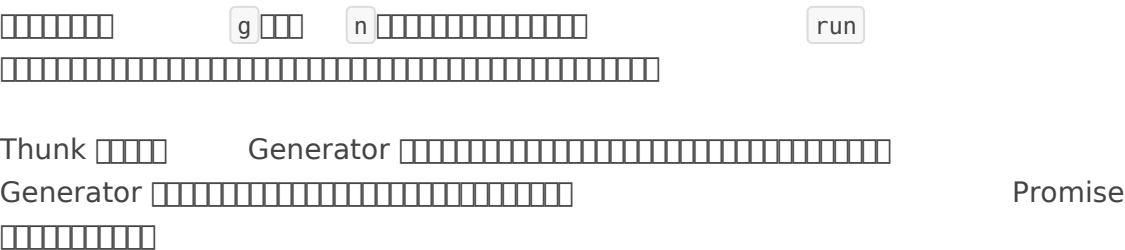
function* g() {
  // ...
}

run(g);
```



```
var g = function* (){
  var f1 = yield readFileThunk('fileA');
  var f2 = yield readFileThunk('fileB');
  // ...
  var fn = yield readFileThunk('fileN');
};

run(g);
```



co



co 1 2 3 4 5 6

TJ Holowaychuk 2013 6

Generator

Generator

```
var gen = function* () {  
  var f1 = yield readFile('/etc/fstab');  
  var f2 = yield readFile('/etc/shells');  
  console.log(f1.toString());  
  console.log(f2.toString());  
};
```

co Generator

```
var co = require('co');  
co(gen);
```

Generator co

co Promise then

```
co(gen).then(function () {  
  console.log('Generator');  
});
```

Generator

CO

co Generator

Generator

1 Think

2 Promise Promise then

☐ ☐ Promise ☐ ☐ ☐ ☐ ☐ ☐

```
var fs = require('fs');

var readFile = function (fileName){
  return new Promise(function (resolve, reject){
    fs.readFile(fileName, function(error, data){
      if (error) return reject(error);
      resolve(data);
    });
  });
};

var gen = function* (){
  var f1 = yield readFile('/etc/fstab');
  var f2 = yield readFile('/etc/shells');
  console.log(f1.toString());
  console.log(f2.toString());
};
```

Generator

```
var g = gen();

g.next().value.then(function(data){
  g.next(data).value.then(function(data){
    g.next(data);
  });
});
```

then

 Generator next

CO

--	--	--	--	--

[illegible]

co Generator Promise

```
function co(gen) {
  var ctx = this;

  return new Promise(function(resolve, reject) {
    });
}
```

Promise ☐ co ☐ gen ☐ Generator

Promise ☐

resolved ☐

```
function co(gen) {
  var ctx = this;

  return new Promise(function(resolve, reject) {
    if (typeof gen === 'function') gen = gen.call(ctx);
```

```

    if (!gen || typeof gen.next !== 'function') return resolve(gen);
  });
}

```

co Generator

next

onFulfilled

```

function co(gen) {
  var ctx = this;

  return new Promise(function(resolve, reject) {
    if (typeof gen === 'function') gen = gen.call(ctx);
    if (!gen || typeof gen.next !== 'function') return resolve(gen);

    onFulfilled();
    function onFulfilled(res) {
      var ret;
      try {
        ret = gen.next(res);
      } catch (e) {
        return reject(e);
      }
      next(ret);
    }
  });
}

```

next

```

function next(ret) {
  if (ret.done) return resolve(ret.value);
  var value = toPromise.call(ctx, ret.value);
  if (value && isPromise(value)) return value.then(onFulfilled, onRejected);
  return onRejected(
    new TypeError(
      'You may only yield a function, promise, generator, array, or object, '
      + 'but the following object was passed: "'
      + String(ret.value)
      + '"
    )
  );
}

```


 Generator 

☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ Promise ☐ ☐

[illegible]

Thunk Promise Promise

rejected

--	--	--	--	--	--	--	--	--

[illegible][illegible]

```
// 正常
co(function* () {
  var res = yield [
    Promise.resolve(1),
    Promise.resolve(2)
  ];
  console.log(res);
}).catch(onerror);

// 异常
co(function* () {
  var res = yield {
    1: Promise.resolve(1),
    2: Promise.resolve(2),
  };
  console.log(res);
}).catch(onerror);
```

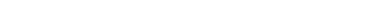
```
co(function* () {
  var values = [n1, n2, n3];
```

```
yield values.map(somethingAsync);
});

function* somethingAsync(x) {
  // do something async
  return y
}
```

The diagram shows a thread executing `somethingAsync`. The timeline consists of 10 ticks. The thread starts at tick 1 and continues to tick 10. The `somethingAsync` function is represented by a box that spans from tick 1 to tick 10. The thread is represented by a vertical line that starts at tick 1 and ends at tick 10. The thread is in a state of `running` from tick 1 to tick 10.

Stream

Node  Stream 
 "  Stream  EventEmitter API 

- data 
- end  "  " 
- error 

```

[] Promise.race() []
[] [] data
[] while []

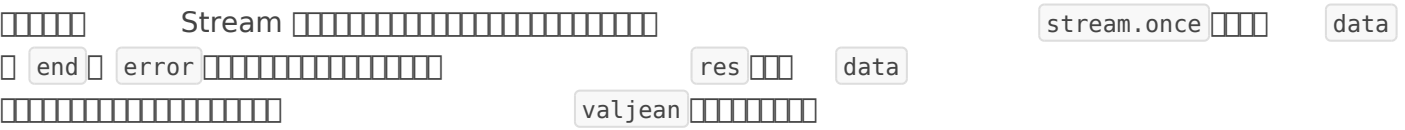
```

```
const co = require('co');
const fs = require('fs');

const stream = fs.createReadStream('./les_miserables.txt');
let valjeanCount = 0;

co(function*() {
  while(true) {
    const res = yield Promise.race([
      new Promise(resolve => stream.once('data', resolve)),
      new Promise(resolve => stream.once('end', resolve)),
      new Promise((resolve, reject) => stream.once('error', reject))
    ]);
    if (!res) {
      break;
    }
    stream.removeAllListeners('data');
    stream.removeAllListeners('end');
    stream.removeAllListeners('error');
```

```
        valjeanCount += (res.toString().match(/valjean/ig) || []).length;
    }
    console.log('count:', valjeanCount); // count: 1120
});
```



```
## #2
# 2020-04-16 10:29:52 UTC
# 2020-04-16 10:29:52 UTC
```