



sonic

Video Cassette Recorder



# Generators

SP LP SLP  
SPEED

SLOW TR

Os generators tornam possível pausar a execução de uma determinada função, permitindo a utilização do event loop de forma **cooperativa**

A screenshot of a Mac OS X desktop environment showing a terminal window and a code editor side-by-side.

The terminal window on the right shows the command:

```
rodrigobranas:javascriptmasterclass $ node generators/generators_1.js
```

The output of the command is displayed below the command:

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30
```

The code editor on the left contains the following JavaScript code:

```
JS generators_1.js x  
generators_1.js — javascriptmasterclass  
  
1 function forever() {  
2     let value = 1;  
3     while (true) {  
4         console.log(value++);  
5     }  
6 }  
7  
8 function today() {  
9     let date = new Date();  
10    console.log(date);  
11 }  
12  
13 forever();  
14 today();  
15
```

A screenshot of a Mac OS X desktop environment showing a terminal window and a code editor. The terminal window is titled 'generators\_2.js — javascriptmasterclass' and contains the command 'node generators/generators\_2.js'. The output shows the date and time '2019-01-20T18:37:58.476Z'. The code editor window is titled 'generators\_2.js' and contains the following JavaScript code:

```
1 function* forever() {
2     let value = 1;
3     while (true) {
4         console.log(value++);
5     }
6 }
7
8 function today() {
9     let date = new Date();
10    console.log(date);
11 }
12
13 forever();
14 today();
15
```

A screenshot of a terminal window titled "generators\_3.js — javascriptmasterclass". The terminal shows the command \$ node generators/generators\_3.js followed by its output: {} object [ 'constructor', 'next', 'return', 'throw' ] 2019-01-20T18:38:08.010Z

```
JS generators_3.js x
generators_3.js — javascriptmasterclass
1 function* forever() {
2     let value = 1;
3     while (true) {
4         console.log(value++);
5     }
6 }
7
8 function today() {
9     let date = new Date();
10    console.log(date);
11 }
12
13 const foreverGenerator = forever();
14 console.log(foreverGenerator);
15 console.log(typeof foreverGenerator);
16 console.log(Object.getOwnPropertyNames(foreverGenerator.__pr
17 today();
18
```

TERMINAL ... 1: bash + =

```
rodrigobranas:javascriptmasterclass $ node generators/generators_3.js
{}
object
[ 'constructor', 'next', 'return', 'throw' ]
2019-01-20T18:38:08.010Z
rodrigobranas:javascriptmasterclass $
```

Os generators utilizam o método **next**  
para iterar sobre os valores disponíveis  
durante a execução da função

generators\_4.js — javascriptmasterclass

JS generators\_4.js x

TERMINAL ... 1: bash + =

```
function* forever() {
    let value = 1;
    while (true) {
        console.log(value++);
    }
}

function today() {
    let date = new Date();
    console.log(date);
}

const foreverGenerator = forever();
foreverGenerator.next();
today();
```

rodrigobranas:javascriptmasterclass \$ node generators/generators\_4.js

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30

Ao encontrar um **yield**, a execução da função é pausada até o método next ser invocado novamente

A screenshot of a Mac OS X desktop environment showing a terminal window and a code editor.

The terminal window is titled "TERMINAL" and has a tab labeled "1: bash". It displays the command "node generators/generators\_5.js" followed by the output "1" and the timestamp "2019-01-20T18:39:12.349Z".

The code editor window is titled "generators\_5.js — javascriptmasterclass". It contains the following JavaScript code:

```
JS generators_5.js x
generators_5.js — javascriptmasterclass

1 function* forever() {
2     let value = 1;
3     while (true) {
4         console.log(value++);
5         yield;
6     }
7 }
8
9 function today() {
10    let date = new Date();
11    console.log(date);
12 }
13
14 const foreverGenerator = forever();
15 foreverGenerator.next();
16 today();
17
```

A screenshot of a Mac OS X desktop environment showing a terminal window and a code editor side-by-side.

The terminal window on the right shows the command `node generators_6.js` being run, followed by the output of the generator function's logs:

```
rodrigobranas:javascriptmasterclass $ node generators/generators_6.js
1
2
3
2019-01-20T18:39:19.163Z
4
5
rodrigobranas:javascriptmasterclass $
```

The code editor window on the left contains the file `generators_6.js` with the following content:

```
JS generators_6.js x
generators_6.js — javascriptmasterclass
1  function* forever() {
2      let value = 1;
3      while (true) {
4          console.log(value++);
5          yield;
6      }
7  }
8
9  function today() {
10     let date = new Date();
11     console.log(date);
12 }
13
14 const foreverGenerator = forever();
15 foreverGenerator.next();
16 foreverGenerator.next();
17 foreverGenerator.next();
18 today();
19 foreverGenerator.next();
20 foreverGenerator.next();
21
```

O retorno do método `next` é um objeto  
contendo value e done, seguindo o  
protocolo de iteração

A screenshot of a Mac OS X desktop environment showing a terminal window and a code editor window.

The terminal window (top right) has the title "TERMINAL" and "1: bash". It displays the output of running the script:

```
rodrigobranas:javascriptmasterclass $ node generators/generators_7.js
1
{ value: undefined, done: false }
2
{ value: undefined, done: false }
3
{ value: undefined, done: false }
2019-01-20T18:39:26.134Z
4
{ value: undefined, done: false }
5
{ value: undefined, done: false }
rodrigobranas:javascriptmasterclass $
```

The code editor window (left side) has the title "generators\_7.js — javascriptmasterclass". It contains the following JavaScript code:

```
JS generators_7.js x
generators_7.js — javascriptmasterclass

1 function* forever() {
2     let value = 1;
3     while (true) {
4         console.log(value++);
5         yield;
6     }
7 }
8
9 function today() {
10    let date = new Date();
11    console.log(date);
12 }
13
14 const foreverGenerator = forever();
15 console.log(foreverGenerator.next());
16 console.log(foreverGenerator.next());
17 console.log(foreverGenerator.next());
18 today();
19 console.log(foreverGenerator.next());
20 console.log(foreverGenerator.next());
21
```

Por meio do **`yield`** é possível retornar  
valores de forma similar ao `return`

A screenshot of a Mac OS X desktop environment showing a terminal window and a code editor.

The terminal window (top right) has a tab labeled "1: bash". It displays the output of running the file "generators\_8.js" with Node.js, showing five iterations of a generator function named "forever" that yields values 1 through 5. The command used was \$ node generators/generators\_8.js.

The code editor window (left) shows the source code for "generators\_8.js". The code defines a generator function "forever" that uses a while loop and a yield statement to return successive values. It also defines a function "today" that logs the current date to the console. The code is part of a larger script that creates a generator object, logs its first two values, and then logs the next three values from the generator, followed by the date from the "today" function, and finally the next two values from the generator.

```
generators_8.js — javascriptmasterclass
JS generators_8.js x
1  function* forever() {
2      let value = 1;
3      while (true) {
4          yield value++;
5      }
6  }
7
8  function today() {
9      let date = new Date();
10     console.log(date);
11 }
12
13 const foreverGenerator = forever();
14 console.log(foreverGenerator.next());
15 console.log(foreverGenerator.next());
16 console.log(foreverGenerator.next());
17 today();
18 console.log(foreverGenerator.next());
19 console.log(foreverGenerator.next());
20
```

```
rodrigobranas:javascriptmasterclass $ node generators/generators_8.js
{ value: 1, done: false }
{ value: 2, done: false }
{ value: 3, done: false }
2019-01-20T18:39:34.245Z
{ value: 4, done: false }
{ value: 5, done: false }
rodrigobranas:javascriptmasterclass $
```

Além disso, também é possível enviar um valor para dentro do generator por meio do método next

A screenshot of a Mac OS X desktop environment showing a terminal window and a code editor window.

The terminal window (top right) has the title "TERMINAL" and "1: bash". It displays the command "node generators/generators\_9.js" followed by several log entries:

```
rodrigobranas:javascriptmasterclass $ node generators/generators_9.js
{ value: 1, done: false }
{ value: 2, done: false }
{ value: 3, done: false }
2019-01-20T18:39:40.641Z
{ value: 1, done: false }
{ value: 2, done: false }
rodrigobranas:javascriptmasterclass $
```

The code editor window (left side) has the title "generators\_9.js" and shows the following JavaScript code:

```
JS generators_9.js x
generators_9.js — javascriptmasterclass

1  function* forever() {
2      let value = 1;
3      while (true) {
4          let reset = yield value++;
5          if (reset) value = 1;
6      }
7  }
8
9  function today() {
10     let date = new Date();
11     console.log(date);
12 }
13
14 const foreverGenerator = forever();
15 console.log(foreverGenerator.next());
16 console.log(foreverGenerator.next());
17 console.log(foreverGenerator.next());
18 today();
19 console.log(foreverGenerator.next(true));
20 console.log(foreverGenerator.next());
```

O método **return** encerra o generator  
podendo retornar um valor específico

A screenshot of a Mac OS X desktop environment showing a terminal window and a code editor side-by-side.

The terminal window on the right shows the output of running the file `generators_10.js` with the command `node generators/generators_10.js`. The output is:

```
rodrigobranas:javascriptmasterclass $ node generators/generators_10.js
{ value: 1, done: false }
{ value: 2, done: false }
{ value: 3, done: false }
2019-01-20T18:39:47.110Z
{ value: undefined, done: true }
{ value: undefined, done: true }
rodrigobranas:javascriptmasterclass $
```

The code editor window on the left contains the file `generators_10.js`:

```
JS generators_10.js x generators_10.js — javascriptmasterclass
1 function* forever() {
2     let value = 1;
3     while (true) {
4         let reset = yield value++;
5         if (reset) value = 1;
6     }
7 }
8
9 function today() {
10    let date = new Date();
11    console.log(date);
12 }
13
14 const foreverGenerator = forever();
15 console.log(foreverGenerator.next());
16 console.log(foreverGenerator.next());
17 console.log(foreverGenerator.next());
18 today();
19 foreverGenerator.return();
20 console.log(foreverGenerator.next(true));
21 console.log(foreverGenerator.next());
```

A screenshot of a Mac OS X desktop environment showing a terminal window and a code editor side-by-side.

The terminal window on the right shows the command `node generators/generators_11.js` being run, followed by the output of three log entries from the generator function, the current date and time, and finally three entries indicating the generator has completed (done: true) and returned "end".

```
generators_11.js — javascriptmasterclass
JS generators_11.js x
1  function* forever() {
2      let value = 1;
3      while (true) {
4          let reset = yield value++;
5          if (reset) value = 1;
6      }
7  }
8
9  function today() {
10     let date = new Date();
11     console.log(date);
12 }
13
14 const foreverGenerator = forever();
15 console.log(foreverGenerator.next());
16 console.log(foreverGenerator.next());
17 console.log(foreverGenerator.next());
18 today();
19 console.log(foreverGenerator.return("end"));
20 console.log(foreverGenerator.next(true));
21 console.log(foreverGenerator.next());
22
```

TERMINAL ... 1: bash + =

```
rodrigobranas:javascriptmasterclass $ node generators/generators_11.js
{ value: 1, done: false }
{ value: 2, done: false }
{ value: 3, done: false }
2019-01-20T18:39:54.098Z
{ value: 'end', done: true }
{ value: undefined, done: true }
{ value: undefined, done: true }
rodrigobranas:javascriptmasterclass $
```

O método **throw** lança uma exceção dentro do generator interrompendo o fluxo de execução caso a exceção não tenha sido tratada adequadamente

A screenshot of a Mac OS X desktop environment showing a terminal window and a code editor side-by-side.

The terminal window on the right shows the output of running the file `generators_12.js` with the command `node generators/generators_12.js`. The output is:

```
rodrigobranas:javascriptmasterclass $ node generators/generators_12.js
{ value: 1, done: false }
{ value: 2, done: false }
{ value: 3, done: false }
2019-01-20T18:40:00.597Z
```

The code editor on the left contains the file `generators_12.js`:

```
JS generators_12.js x
generators_12.js — javascriptmasterclass
1 function* forever() {
2     let value = 1;
3     while (true) {
4         let reset = yield value++;
5         if (reset) value = 1;
6     }
7 }
8
9 function today() {
10    let date = new Date();
11    console.log(date);
12 }
13
14 const foreverGenerator = forever();
15 console.log(foreverGenerator.next());
16 console.log(foreverGenerator.next());
17 console.log(foreverGenerator.next());
18 today();
19 console.log(foreverGenerator.throw("error"));
20 console.log(foreverGenerator.next(true));
21 console.log(foreverGenerator.next());
```

A screenshot of a Mac OS X desktop environment showing a terminal window and a code editor side-by-side.

The terminal window on the right shows the output of running a JavaScript file named `generators_13.js` using the command `node generators/generators_13.js`. The output is as follows:

```
rodrigobranas:javascriptmasterclass $ node generators/generators_13.js
{ value: 1, done: false }
{ value: 2, done: false }
{ value: 3, done: false }
2019-01-20T18:40:16.296Z
error
{ value: 4, done: false }
{ value: 1, done: false }
{ value: 2, done: false }
rodrigobranas:javascriptmasterclass $
```

The code editor on the left contains the following JavaScript code:

```
JS generators_13.js x
generators_13.js — javascriptmasterclass
1 function* forever() {
2     let value = 1;
3     while (true) {
4         try {
5             let reset = yield value++;
6             if (reset) value = 1;
7         } catch (e) {
8             console.log(e);
9         }
10    }
11 }
12
13 function today() {
14     let date = new Date();
15     console.log(date);
16 }
17
18 const foreverGenerator = forever();
19 console.log(foreverGenerator.next());
20 console.log(foreverGenerator.next());
21 console.log(foreverGenerator.next());
22 today();
23 console.log(foreverGenerator.throw("error"));
24 console.log(foreverGenerator.next(true));
25 console.log(foreverGenerator.next());
```



Onde é possível utilizar os generators?

Como os generators **implementam** o  
**protocolo de iteração** é possível utilizá-los  
com `Symbol.iterator` de forma simples

A screenshot of a Mac OS X desktop environment showing a terminal window and a code editor. The terminal window is titled '1: bash' and contains the command 'node generators/generators\_14.js' followed by the output 'Fortran', 'Lisp', and 'COBOL'. The code editor window is titled 'generators\_14.js — javascriptmasterclass' and shows the source code for a generator function named 'createIterable'. The code defines a generator that returns an iterator object with a 'next' method. The 'next' method checks if the current index 'i' is less than the array length. If true, it returns an object with 'value' set to the current element and 'done' set to false. If false, it returns an object with 'value' set to undefined and 'done' set to true. The code then logs the language names to the console.

```
JS generators_14.js x
generators_14.js — javascriptmasterclass
TERMINAL ... 1: bash + = 1
rodrigobranas:javascriptmasterclass $ node generators/generators_14.js
Fortran
Lisp
COBOL
rodrigobranas:javascriptmasterclass $
```

```
1  function createIterable(...array) {
2    return {
3      [Symbol.iterator]() {
4        let i = 0;
5        return {
6          next() {
7            if (i < array.length) {
8              return {
9                value: array[i++],
10               done: false
11             }
12           } else {
13             return {
14               value: undefined,
15               done: true
16             }
17           }
18         }
19       }
20     }
21   }
22 }
23 const iterable = createIterable("Fortran", "Lisp", "COBOL");
24 for (let language of iterable) {
25   console.log(language);
26 }
```

A screenshot of a Mac OS X desktop environment showing a terminal window and a code editor.

The terminal window (top right) has a title bar "TERMINAL" and a dropdown menu "1: bash". The command "node generators/generators\_15.js" is run, followed by the output:

```
rodrigobranas:javascriptmasterclass $ node generators/generators_15.js
Fortran
Lisp
COBOL
rodrigobranas:javascriptmasterclass $
```

The code editor (left side) shows a file named "generators\_15.js" with the following content:

```
1  function createIterable(...array) {
2      return {
3          *[Symbol.iterator]() {
4              let i = 0;
5              while (i < array.length) {
6                  yield array[i++];
7              }
8          }
9      }
10 }
11 const iterable = createIterable("Fortran", "Lisp", "COBOL");
12 for (let language of iterable) {
13     console.log(language);
14 }
15
```

Além disso, é possível utilizar generators para **sincronizar chamadas assíncronas** de forma similar ao `async/await`

A screenshot of a Mac OS X desktop environment showing a terminal window and a code editor window.

The terminal window (top right) has the title "TERMINAL" and shows the command:

```
rodrigobranas:javascriptmasterclass $ node generators/generators_16.js
```

The output of the command is:

```
12
```

The code editor window (left side) has the title "generators\_16.js — javascriptmasterclass". It contains the following JavaScript code:1 function sum(a, b) {  
2 return new Promise(function (resolve) {  
3 setTimeout(function() {  
4 resolve(a + b);  
5 }, 1000);  
6 });  
7 }  
8 sum(2, 2).then(function(a) {  
9 sum(4, 4).then(function(b) {  
10 sum(a, b).then(function(result) {  
11 console.log(result);  
12 });  
13 });  
14});  
15

A screenshot of a Mac OS X desktop environment showing a terminal window and a code editor.

The terminal window (top right) has a tab labeled "1: bash". It displays the command:

```
rodrigobranas:javascriptmasterclass $ node generators/generators_17.js
```

The code editor window (left) has a tab labeled "generators\_17.js". It contains the following JavaScript code:

```
function sum(a, b) {
  return new Promise(function (resolve) {
    setTimeout(function() {
      resolve(a + b);
    }, 1000);
  });
}

const a = sum(2, 2);
const b = sum(4, 4)
const result = sum(a, b);
console.log(result);
```

A screenshot of a Mac OS X desktop environment showing a terminal window and a code editor window.

The terminal window (top right) shows the command:

```
rodrigobranas:javascriptmasterclass $ node generators/generators_18.js
```

The output of the command is:

```
12
```

The code editor window (left side) contains the file `generators_18.js` with the following content:

```
1  function sum(a, b) {
2      return new Promise(function (resolve) {
3          setTimeout(function() {
4              resolve(a + b);
5          }, 1000);
6      });
7  }
8  function async(fn) {
9      const gen = fn();
10     asyncR(gen);
11 }
12 function asyncR(gen, value) {
13     const obj = gen.next(value);
14     if (obj.done) return;
15     obj.value.then(function (result) {
16         asyncR(gen, result);
17     });
18 }
19 async(function* () {
20     const a = yield sum(2, 2);
21     const b = yield sum(4, 4)
22     const result = yield sum(a, b);
23     console.log(result);
24 });
25 }
```