

A photograph of a large stack of shipping containers under a clear blue sky. The containers are stacked in several layers, creating a vertical pattern of colors. Visible colors include orange, red, yellow, blue, green, and white. Some containers have doors or markings. The perspective is from the side, looking along the length of the stack.

Módulos



A linguagem JavaScript nasceu dentro do navegador e uma das características da Web era justamente o conceito de **Code on Demand** e isso era realizado pela tag <SCRIPT>

Como a linguagem oficialmente não tinha suporte a qualquer sistema de módulos, algumas abordagens foram criadas, entre elas o **CommonJS** ou CJS

Ele nasceu na Mozilla em 2009 com o nome de
ServerJS, sendo renomeado alguns meses depois
para CommonJS

Nele, existe uma **relação direta entre um módulo de um arquivo, que pode ter parte do seu conteúdo exportado e importado em outro módulo**



Basicamente é só usar `module.exports` para exportar e `require` para importar

The screenshot shows a macOS-style application window with two tabs open:

- main.js**: This tab is currently inactive.
- generator.js**: This tab is active, indicated by a grey background and a white tab indicator.

The content of the active tab, `generator.js`, is as follows:

```
1 const max = 100;
2
3 const generate = function () {
4     return Math.floor(Math.random() * max);
5 }
6
```

The code defines a constant `max` set to 100, and a function `generate` that returns a random integer between 0 and `max` (inclusive) using the `Math.floor` and `Math.random` methods.

The screenshot shows a code editor window with a tab bar at the top containing "main.js" and "generator.js". The "main.js" file is open and contains the following code:

```
1 const generator = require("./generator");
2 console.log(generator);
```

The "generator.js" file is empty. Below the editor is a navigation bar with tabs for "PROBLEMS", "OUTPUT", "TERMINAL", and "DEBUG CONSOLE". The "TERMINAL" tab is selected, showing the command "node main.js" and its output: "[]".

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

rodrigobranas nodejs \$ node main.js
[]
rodrigobranas nodejs \$



Por que o módulo está **vazio**?



Tudo que é definido dentro do módulo
é **privado**

The screenshot shows a macOS-style application window with a title bar "generator.js — nodejs". There are two tabs visible: "main.js" and "generator.js". The "generator.js" tab is currently selected. The code in the editor is as follows:

```
1 const max = 100;
2
3 module.exports.generate = function () {
4     ...
5     return Math.floor(Math.random() * max);
6 }
```

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows two files: `main.js` and `generator.js`. `main.js` is the active file.
- Code Editor:** The `main.js` file contains the following code:

```
1 const generator = require("./generator");
2 console.log(generator);
```
- Terminal:** The terminal tab is active, showing the command `node main.js` and its output:

```
rodrigobranas nodejs $ node main.js
{ generate: [Function (anonymous)] }
rodrigobranas nodejs $ █
```
- Bottom Bar:** Includes tabs for PROBLEMS, OUTPUT, TERMINAL (underlined), and DEBUG CONSOLE. It also features a terminal icon with "zsh" and other icons for switching, closing, and opening multiple terminals.

A screenshot of a Mac OS X terminal window titled "main.js — nodejs". The window contains two tabs: "main.js" and "generator.js". The "generator.js" tab is active, showing the following code:

```
1 const generator = require("./generator");
2 console.log(generator.generate());
```

The terminal pane below shows the output of running the "main.js" script:

```
rodrigobranas nodejs $ node main.js
98
rodrigobranas nodejs $ █
```



Também é possível exportar o módulo
utilizando `exports` ou `this`

The screenshot shows a macOS-style code editor with a dark theme. There are two tabs open: "main.js" and "generator.js". The "generator.js" tab is currently active, showing the following code:

```
1 const max = 100;
2
3 exports.generate = function () {
4     return Math.floor(Math.random() * max);
5 }
6
```

Below the editor is a terminal window with the following output:

```
rodrigobranas nodejs $ node main.js
{ generate: [Function (anonymous)] }
rodrigobranas nodejs $ █
```

A screenshot of a macOS terminal window titled "generator.js — nodejs". The window has two tabs: "main.js" (selected) and "generator.js X". The "main.js" tab contains the following code:

```
1 const max = 100;
2
3 this.generate = function () {
4     return Math.floor(Math.random() * max);
5 }
6
```

The "generator.js" tab is empty.

Below the tabs, there are navigation icons: a red circle, a yellow circle, a green circle, a close button, a maximize button, and a menu icon.

At the bottom, there are tabs for "PROBLEMS", "OUTPUT", "TERMINAL" (underlined), and "DEBUG CONSOLE". To the right of these tabs is a terminal interface with the following history:

```
rodrigobranas nodejs $ node main.js
{ generate: [Function (anonymous)] }
rodrigobranas nodejs $ █
```

The terminal interface includes icons for opening a new tab ("zsh +"), closing the tab ("X"), and other terminal-specific functions.



Qual é a diferença entre
module.exports, **exports** e **this**?

The screenshot shows a macOS-style code editor window with the following details:

- Title Bar:** "main.js — nodejs" is displayed at the top center.
- File Tabs:** Two tabs are visible: "JS main.js" (active) and "JS generator.js".
- Code Editor:** The "main.js" tab contains the following code:

```
1 console.log(module.exports === exports);
2 console.log(exports === this);
3 console.log(module.exports === this);
4
```
- Terminal:** The "TERMINAL" tab is active, showing the command "node main.js" and its output:

```
rodrigobranas nodejs $ node main.js
true
true
true
rodrigobranas nodejs $ █
```
- Toolbar:** A toolbar at the bottom right includes icons for zsh, terminal navigation, and other utilities.



CAUTION

Cuidado, apenas `module.exports`
é retornado da função `require`

node-v0.x-archive/node.js a... +

GitHub, Inc. (US) https://github.com/nodejs/node-v0.x-archive/blob/master/src/node.js Pesquisar

This repository Search Pull requests Issues Gist

nodejs / node-v0.x-archive Watch 2,546 Star 37,415 Fork 8,443

Code Issues 597 Pull requests 43 Wiki Pulse Graphs

Branch: master node-v0.x-archive / src / node.js Find file Copy path

misterdjules src: enable strict mode in all builtin modules ef43443 on 3 Mar 2015

63 contributors and others

819 lines (665 sloc) | 23.3 KB Raw Blame History

```
1 // Copyright Joyent, Inc. and other Node contributors.  
2 //  
3 // Permission is hereby granted, free of charge, to any person obtaining a  
4 // copy of this software and associated documentation files (the  
5 // "Software"), to deal in the Software without restriction, including  
6 // without limitation the rights to use, copy, modify, merge, publish,  
7 // distribute, sublicense, and/or sell copies of the Software, and to permit  
8 // persons to whom the Software is furnished to do so, subject to the  
9 // following conditions:  
10 //  
11 // The above copyright notice and this permission notice shall be included  
12 // in all copies or substantial portions of the Software.  
13 //  
14 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS  
15 // OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF  
16 // MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN  
17 // NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,  
18 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR  
19 // OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE  
20 // USE OR OTHER DEALINGS IN THE SOFTWARE.  
21 //  
22 // Hello, and welcome to hacking node.js!  
23 //  
24 // This file is invoked by node::Load in src/node.cc, and responsible for  
25 // bootstrapping the node.js core. Special caution is given to the performance
```

<https://github.com/nodejs/node-v0.x-archive/blob/master/src/node.js>

```
748 function NativeModule(id) {
749     this.filename = id + '.js';
750     this.id = id;
751     this.exports = {};
752     this.loaded = false;
753 }
754
755 NativeModule._source = process.binding('natives');
756 NativeModule._cache = {};
757
758 NativeModule.require = function(id) {
759     if (id == 'native_module') {
760         return NativeModule;
761     }
762
763     var cached = NativeModule.getCached(id);
764     if (cached) {
765         return cached.exports;
766     }
767
768     if (!NativeModule.exists(id)) {
769         throw new Error('No such native module ' + id);
770     }
771
772     process.moduleLoadList.push('NativeModule ' + id);
773
774     var nativeModule = new NativeModule(id);
775
776     nativeModule.cache();
777     nativeModule.compile();
778
779     return nativeModule.exports;
780 };
```

```
794 NativeModule.wrap = function(script) {
795   return NativeModule.wrapper[0] + script + NativeModule.wrapper[1];
796 }
797
798 NativeModule.wrapper = [
799   '(function (exports, require, module, __filename, __dirname) { ',
800   '\n});'
801 ];
802
803 NativeModule.prototype.compile = function() {
804   var source = NativeModule.getSource(this.id);
805   source = NativeModule.wrap(source);
806
807   var fn = runInThisContext(source, { filename: this.filename });
808   fn(this.exports, NativeModule.require, this, this.filename);
809
810   this.loaded = true;
811 };
812
```

A screenshot of a Mac OS X terminal window titled "main.js — nodejs". The window has a tab bar with two tabs: "main.js" and "generator.js". The "main.js" tab is active, showing the code "1 console.log(arguments);". Below the tabs is a toolbar with icons for PROBLEMS, OUTPUT, TERMINAL (which is underlined), and DEBUG CONSOLE. To the right of the toolbar are icons for switching between panes, closing the window, and other system functions. The main pane displays the output of running the script, which shows the module loading process. The output is as follows:

```
path: '/Users/rodrigobranas/development/workspace/branas/fullstackjs/nodejs',
exports: {},
filename: '/Users/rodrigobranas/development/workspace/branas/fullstackjs/nodejs/main.js',
loaded: false,
children: [],
paths: [
  '/Users/rodrigobranas/development/workspace/branas/fullstackjs/nodejs/node_modules',
  '/Users/rodrigobranas/development/workspace/branas/fullstackjs/node_modules',
  '/Users/rodrigobranas/development/workspace/branas/node_modules',
  '/Users/rodrigobranas/development/workspace/node_modules',
  '/Users/rodrigobranas/development/node_modules',
  '/Users/rodrigobranas/node_modules',
  '/Users/node_modules',
  '/node_modules'
],
},
'3': '/Users/rodrigobranas/development/workspace/branas/fullstackjs/nodejs/main.js',
'4': '/Users/rodrigobranas/development/workspace/branas/fullstackjs/nodejs'
}
rodrigobranas nodejs $
```

Como um módulo é localizado?



Primeiro, o algoritmo de busca tenta localizar um módulo **core**. Existem vários como: net, http, url, fs, zlib, crypto, events, stream, os, vm, util, entre outros.

`require(X)` from module at path Y

1. If X is a core module,
 - a. `return` the core module
 - b. STOP
2. If X begins with `'./'` or `'/'` or `'../'`
 - a. `LOAD_AS_FILE(Y + X)`
 - b. `LOAD_AS_DIRECTORY(Y + X)`
3. `LOAD_NODE_MODULES(X, dirname(Y))`
4. `THROW "not found"`

`LOAD_AS_FILE(X)`

1. If X is a file, load X as JavaScript text. STOP
2. If X.js is a file, load X.js as JavaScript text. STOP
3. If X.json is a file, parse X.json to a JavaScript Object. STOP
4. If X.node is a file, load X.node as binary addon. STOP

`LOAD_AS_DIRECTORY(X)`

1. If X/package.json is a file,
 - a. Parse X/package.json, and look for `"main"` field.
 - b. let M = X + (json main field)
 - c. `LOAD_AS_FILE(M)`
2. If X/index.js is a file, load X/index.js as JavaScript text. STOP
3. If X/index.json is a file, parse X/index.json to a JavaScript object. STOP
4. If X/index.node is a file, load X/index.node as binary addon. STOP

A screenshot of a macOS terminal window titled "main.js — nodejs". The window contains a code editor with a single file named "main.js" containing the following code:

```
1 const http = require("http");
2 console.log(http);
```

The terminal output shows the execution of the code, displaying the Node.js http module's exports:

```
ClientRequest: [Function: ClientRequest],
IncomingMessage: [Function: IncomingMessage],
OutgoingMessage: [Function: OutgoingMessage],
Server: [Function: Server],
ServerResponse: [Function: ServerResponse],
createServer: [Function: createServer],
validateHeaderName: [Function: __node_internal_],
validateHeaderValue: [Function: __node_internal_],
get: [Function: get],
request: [Function: request],
maxHeaderSize: [Getter],
globalAgent: [Getter/Setter]
}
rodrigobranas nodejs $
```

Se o nome do módulo iniciar com '/', '../' ou './', o algoritmo de busca irá localizar o módulo por meio do caminho absoluto no sistema de arquivos.

`require(X)` from module at path Y

1. If X is a core module,
 - a. `return` the core module
 - b. STOP

2. If X begins `'./'` or `'/'` or `'../'`

- a. `LOAD_AS_FILE(Y + X)`
- b. `LOAD_AS_DIRECTORY(Y + X)`

3. `LOAD_NODE_MODULES(X, dirname(Y))`

4. `THROW "not found"`

`LOAD_AS_FILE(X)`

1. If X is a file, load X as JavaScript text. STOP
2. If X.js is a file, load X.js as JavaScript text. STOP
3. If X.json is a file, parse X.json to a JavaScript Object. STOP
4. If X.node is a file, load X.node as binary addon. STOP

`LOAD_AS_DIRECTORY(X)`

1. If X/package.json is a file,
 - a. Parse X/package.json, and look for `"main"` field.
 - b. let M = X + (json main field)
 - c. `LOAD_AS_FILE(M)`
2. If X/index.js is a file, load X/index.js as JavaScript text. STOP
3. If X/index.json is a file, parse X/index.json to a JavaScript object. STOP
4. If X/index.node is a file, load X/index.node as binary addon. STOP

A screenshot of a Mac OS X window showing a terminal session in the Visual Studio Code editor. The window title is "main.js — nodejs". The code editor tab bar shows "JS main.js" and an "X" button. The code in "main.js" is:

```
1 const generator = require("./generator");
2 console.log(generator.generate());
```

The terminal tab is active, showing the command "node main.js" and its output "98". The terminal interface includes tabs for PROBLEMS, OUTPUT, TERMINAL (underlined), and DEBUG CONSOLE, along with icons for switching between terminals.

Localizando um módulo na mesma pasta



A screenshot of a Mac OS X terminal window titled "main.js — nodejs". The window contains a code editor with a single file named "main.js" and a terminal pane below it.

The code in "main.js" is:

```
1 const generator = require("/Users/rodrigobranas/development/workspace/branas/
2 fullstackjs/nodejs/generator.js");
3 console.log(generator.generate());
```

The terminal pane shows the following output:

```
rodrigobranas nodejs $ node main.js
84
rodrigobranas nodejs $ █
```

Localizando um módulo por meio do caminho absoluto

The screenshot shows a VS Code interface with the following details:

- EXPLORER** sidebar: Shows a tree view of a project named "NODEJS". It contains a folder "a" which has files "main.js" (selected), "config.js", and "generator.js". It also contains a folder "b" and a "package.json" file.
- EDITOR**: A tab bar with "main.js" (highlighted) and "generator.js". The code in "main.js" is:

```
1 const generator = require("../b/generator");
2 console.log(generator.generate());
```
- TERMINAL**: Shows the command "node a/main.js" being run, followed by the output "80".

Localizando um módulo por meio do caminho relativo

Terminar o nome com '.js' é opcional, o algoritmo de busca irá tentar colocar o '.js' no final caso não encontre o módulo.

`require(X)` from module at path Y

1. If X is a core module,
 - a. `return` the core module
 - b. STOP
2. If X begins `'./'` or `'/'` or `'../'`
 - a. `LOAD_AS_FILE(Y + X)`
 - b. `LOAD_AS_DIRECTORY(Y + X)`
3. `LOAD_NODE_MODULES(X, dirname(Y))`
4. `THROW "not found"`

`LOAD_AS_FILE(X)`

1. If X is a file, load X as JavaScript text. STOP
2. If X.js is a file, load X.js as JavaScript text. STOP
3. If X.json is a file, parse X.json to a JavaScript Object. STOP
4. If X.node is a file, load X.node as binary addon. STOP

`LOAD_AS_DIRECTORY(X)`

1. If X/package.json is a file,
 - a. Parse X/package.json, and look for `"main"` field.
 - b. let M = X + (json main field)
 - c. `LOAD_AS_FILE(M)`
2. If X/index.js is a file, load X/index.js as JavaScript text. STOP
3. If X/index.json is a file, parse X/index.json to a JavaScript object. STOP
4. If X/index.node is a file, load X/index.node as binary addon. STOP

Se o módulo não começar com '/', '../' ou './' o algoritmo de busca da função require vai procurar dentro da pasta `node_modules`.

`require(X)` from module at path Y

1. If X is a core module,
 - a. `return` the core module
 - b. STOP
2. If X begins `'./'` or `'/'` or `'../'`
 - a. `LOAD_AS_FILE(Y + X)`
 - b. `LOAD_AS_DIRECTORY(Y + X)`
3. `LOAD_NODE_MODULES(X, dirname(Y))`
4. `THROW "not found"`

`LOAD_AS_FILE(X)`

1. If X is a file, load X as JavaScript text. STOP
2. If X.js is a file, load X.js as JavaScript text. STOP
3. If X.json is a file, parse X.json to a JavaScript Object. STOP
4. If X.node is a file, load X.node as binary addon. STOP

`LOAD_AS_DIRECTORY(X)`

1. If X/package.json is a file,
 - a. Parse X/package.json, and look for `"main"` field.
 - b. let M = X + (json main field)
 - c. `LOAD_AS_FILE(M)`
2. If X/index.js is a file, load X/index.js as JavaScript text. STOP
3. If X/index.json is a file, parse X/index.json to a JavaScript object. STOP
4. If X/index.node is a file, load X/index.node as binary addon. STOP

LOAD_NODE_MODULES(X, START)

1. let DIRS=NODE_MODULES_PATHS(START)
2. for each DIR in DIRS:
 - a. LOAD_AS_FILE(DIR/X)
 - b. LOAD_AS_DIRECTORY(DIR/X)

NODE_MODULES_PATHS(START)

1. let PARTS = path split(START)
2. let I = count of PARTS - 1
3. let DIRS = []
4. while I >= 0,
 - a. if PARTS[I] = "node_modules" CONTINUE
 - c. DIR = path join(PARTS[0 .. I] + "node_modules")
 - b. DIRS = DIRS + DIR
 - c. let I = I - 1
5. return DIRS

A screenshot of a Mac OS X terminal window titled "main.js — nodejs". The window has a tab bar with two tabs: "main.js" and "generator.js". The "main.js" tab is active, showing the code "1 console.log(arguments);". Below the tabs is a toolbar with icons for PROBLEMS, OUTPUT, TERMINAL (which is underlined), and DEBUG CONSOLE. To the right of the toolbar are icons for switching between panes, closing the window, and other system functions. The main content area displays the output of the Node.js process. It shows the module loading process for "main.js", listing the path, exports, filename, loaded status, children, and paths. The paths listed include the current directory and standard Node.js modules like "node_modules". The output ends with the prompt "rodrigobranas nodejs \$".

```
path: '/Users/rodrigobranas/development/workspace/branas/fullstackjs/nodejs',
exports: {},
filename: '/Users/rodrigobranas/development/workspace/branas/fullstackjs/nodejs/main.js',
loaded: false,
children: [],
paths: [
  '/Users/rodrigobranas/development/workspace/branas/fullstackjs/nodejs/node_modules',
  '/Users/rodrigobranas/development/workspace/branas/fullstackjs/node_modules',
  '/Users/rodrigobranas/development/workspace/branas/node_modules',
  '/Users/rodrigobranas/development/workspace/node_modules',
  '/Users/rodrigobranas/development/node_modules',
  '/Users/rodrigobranas/node_modules',
  '/Users/node_modules',
  '/node_modules'
],
},
'3': '/Users/rodrigobranas/development/workspace/branas/fullstackjs/nodejs/main.js',
'4': '/Users/rodrigobranas/development/workspace/branas/fullstackjs/nodejs'
}
rodrigobranas nodejs $
```

A close-up photograph of a person's hands reaching towards a large, clear crystal ball. The hands are positioned as if they are about to touch or lift the sphere. The background is dark and out of focus, making the bright surface of the crystal ball stand out.

A função require utiliza um
mecanismo de **cache**

node-v0.x-archive/node.js a... +

GitHub, Inc. (US) https://github.com/nodejs/node-v0.x-archive/blob/master/src/node.js Pesquisar

This repository Search Pull requests Issues Gist

nodejs / node-v0.x-archive Watch 2,546 Star 37,415 Fork 8,443

Code Issues 597 Pull requests 43 Wiki Pulse Graphs

Branch: master node-v0.x-archive / src / node.js Find file Copy path

misterdjules src: enable strict mode in all builtin modules ef43443 on 3 Mar 2015

63 contributors and others

819 lines (665 sloc) | 23.3 KB Raw Blame History

```
1 // Copyright Joyent, Inc. and other Node contributors.  
2 //  
3 // Permission is hereby granted, free of charge, to any person obtaining a  
4 // copy of this software and associated documentation files (the  
5 // "Software"), to deal in the Software without restriction, including  
6 // without limitation the rights to use, copy, modify, merge, publish,  
7 // distribute, sublicense, and/or sell copies of the Software, and to permit  
8 // persons to whom the Software is furnished to do so, subject to the  
9 // following conditions:  
10 //  
11 // The above copyright notice and this permission notice shall be included  
12 // in all copies or substantial portions of the Software.  
13 //  
14 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS  
15 // OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF  
16 // MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN  
17 // NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,  
18 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR  
19 // OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE  
20 // USE OR OTHER DEALINGS IN THE SOFTWARE.  
21 //  
22 // Hello, and welcome to hacking node.js!  
23 //  
24 // This file is invoked by node::Load in src/node.cc, and responsible for  
25 // bootstrapping the node.js core. Special caution is given to the performance
```

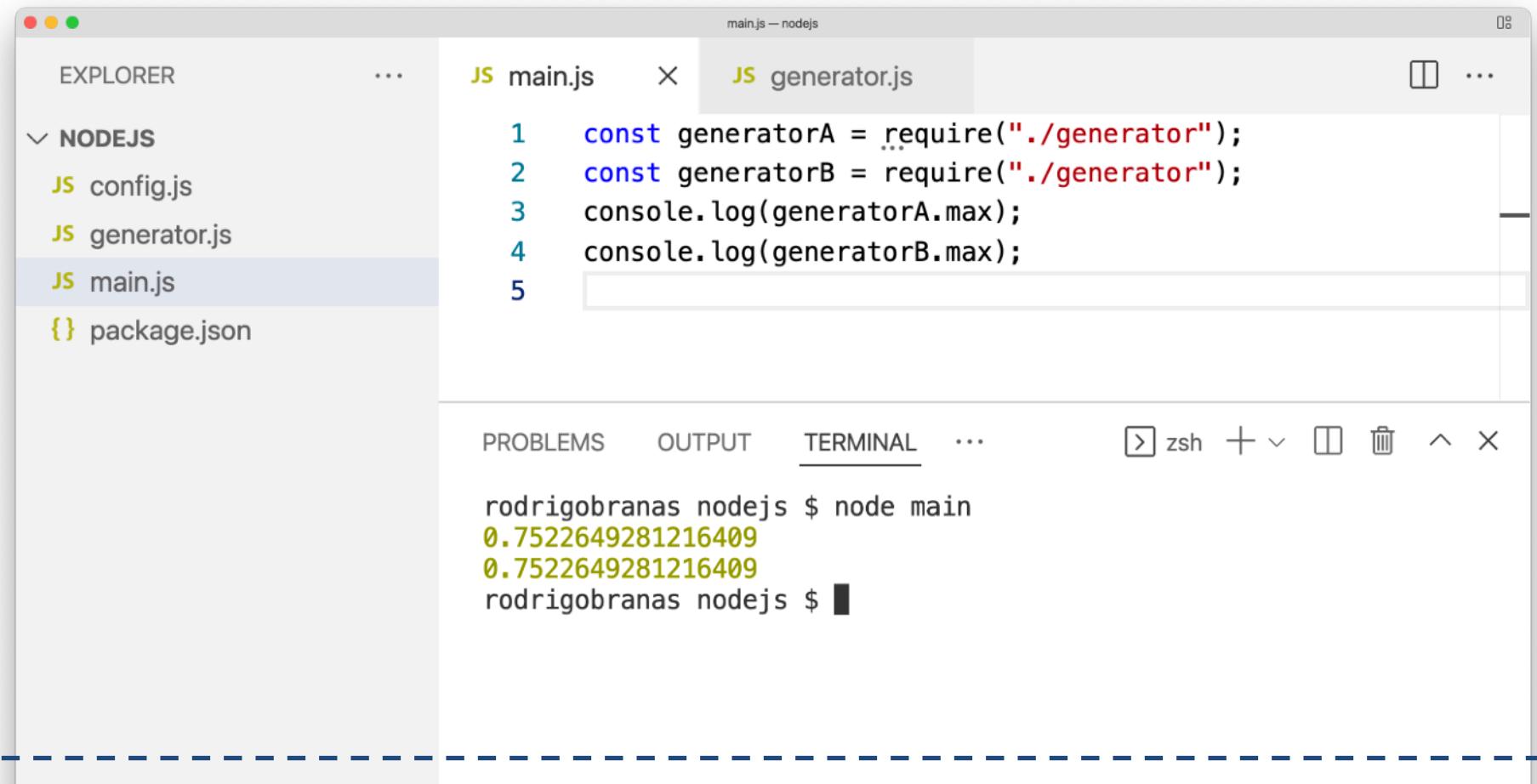
<https://github.com/nodejs/node-v0.x-archive/blob/master/src/node.js>

```
758 NativeModule.require = function(id) {
759   if (id == 'native_module') {
760     return NativeModule;
761   }
762
763   var cached = NativeModule.getCached(id);
764   if (cached) {
765     return cached.exports;
766   }
767
768   if (!NativeModule.exists(id)) {
769     throw new Error('No such native module ' + id);
770   }
771
772   process.moduleLoadList.push('NativeModule ' + id);
773
774   var nativeModule = new NativeModule(id);
775
776   nativeModule.cache();
777   nativeModule.compile();
778
779   return nativeModule.exports;
780 };
```

The screenshot shows a code editor interface with two tabs: "main.js" and "generator.js". The "generator.js" tab is active, displaying the following code:

```
1  this.max = Math.random();
2
3  this.generate = function () {
4      return Math.floor(Math.random() * this.max);
5
6
```

The code defines a generator function named "generate" that returns a random integer between 0 and "this.max". The "main.js" file is also visible in the background.



The screenshot shows a VS Code interface with the following details:

- EXPLORER** sidebar: Shows a tree view with a **NODEJS** folder containing `config.js`, `generator.js`, `main.js`, and `package.json`.
- main.js — nodejs** tab: Active tab, showing the following code:

```
1 const generatorA = require("./generator");
2 const generatorB = require("./generator");
3 console.log(generatorA.max);
4 console.log(generatorB.max);
5 
```
- TERMINAL**: Shows the command `node main` being run, resulting in the output:

```
rodrigobranas nodejs $ node main
0.7522649281216409
0.7522649281216409
rodrigobranas nodejs $ █
```

O módulo serialGenerator ficou
armazenado em cache

Há alguns anos o Node.js dá suporte para os **ES Modules**, começando com a extensão .mjs e agora permitindo a configuração no package.json por meio da propriedade type e o valor module

A screenshot of a code editor showing a Node.js project structure. The left sidebar shows files: generator.js, main.js, and package.json. The package.json file is currently selected and open in the main editor area. The code is a standard package.json object with the following content:

```
{ "name": "nodejs", "version": "1.0.0", "description": "", "main": "index.js", "scripts": { "test": "echo \\\"Error: no test specified\\\" && exit 1" }, "keywords": [], "author": "", "license": "ISC", "type": "module" }
```

The screenshot shows a Mac OS X-style interface for a code editor, likely Visual Studio Code. The title bar indicates the file is named "main.js — nodejs".

The Explorer sidebar on the left shows a folder named "NODEJS" containing files: "generator.js", "main.js", and "package.json".

The main editor area displays the contents of "main.js":

```
1 import generate from "./generator.js";
2 console.log(generate());
3
```

The bottom right corner of the editor has a small preview window showing the current file content.

The bottom navigation bar includes tabs for "PROBLEMS", "OUTPUT", and "TERMINAL". The "TERMINAL" tab is active, showing the command-line interface:

```
rodrigobranas nodejs $ node main.js
78
rodrigobranas nodejs $
```

The bottom left corner of the interface has a "OUTLINE" button.

A screenshot of a code editor showing a Node.js project structure. The left sidebar shows a tree view under 'NODEJS' with files: 'generator.js', 'main.js', and 'package.json'. The main editor area has tabs for 'main.js', 'generator.js', and 'package.json'. The 'generator.js' tab is active, displaying the following code:

```
1 export default function generate () {
2     return Math.floor(Math.random() * 100);
3 }
4
```

A 3D rendering of a space station in orbit around Earth. The station has a central cylindrical module and two solar panel arrays. It is surrounded by various floating objects, including a red balloon, a wrench, and several small spheres of different colors (red, yellow, green). The background shows the blue and white atmosphere of Earth.

Global Objects



Tudo que é definido dentro do módulo
por padrão é **privado**

A screenshot of a code editor interface showing a Node.js project structure. The left sidebar displays the file tree under the 'NODEJS' category:

- generator.js
- main.js
- package.json

The main editor area has two tabs open: 'main.js' and 'generator.js'. The 'generator.js' tab is active, showing the following code:

```
1 module.exports.generate = function () {
2     return Math.floor(Math.random() * max);
3 }
4 
```

A screenshot of the Visual Studio Code (VS Code) interface. The left sidebar shows a file tree with a nodejs folder containing generator.js, main.js, and package.json. The main editor area has two tabs: main.js and generator.js. The generator.js tab is active, displaying the following code:

```
1 const max = 100;
2 const generator = require("./generator");
3 console.log(generator.generate());
```

The bottom panel shows the terminal tab with the following error output:

```
ReferenceError: max is not defined
    at Object.module.exports.generate (/Users/rodrigobranas/development/workspace/branas/fullstackjs/nodejs/generator.js:2:36)
    at Object.<anonymous> (/Users/rodrigobranas/development/workspace/branas/fullstackjs/nodejs/main.js:3:23)
    at Module._compile (node:internal/modules/cjs/loader:1092:14)
    at Object.Module._extensions..js (node:internal/modules/cjs/loader:1121:10)
    at Module.load (node:internal/modules/cjs/loader:972:32)
    at Function.Module._load (node:internal/modules/cjs/loader:813:14)
    at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:76:12)
    at node:internal/main/run_main_module:17:47
rodrigobranas nodejs $
```



Existe um **escopo global**, similar
ao dos navegadores?



Cuidado, evite **poluir** o escopo global



nodejs — node — 81x24

```
[rodrigobranas nodejs $ node
Welcome to Node.js v15.14.0.
Type ".help" for more information.
[> global
<ref *1> Object [global] {
  global: [Circular *1],
  clearInterval: [Function: clearInterval],
  clearTimeout: [Function: clearTimeout],
  setInterval: [Function: setInterval],
  setTimeout: [Function: setTimeout] {
    [Symbol(nodejs.util.promisify.custom)]: [Getter]
  },
  queueMicrotask: [Function: queueMicrotask],
  clearImmediate: [Function: clearImmediate],
  setImmediate: [Function: setImmediate] {
    [Symbol(nodejs.util.promisify.custom)]: [Getter]
  }
}
> ]
```



nodejs — node — 81x24

```
[rodrigobranas nodejs $ node
Welcome to Node.js v15.14.0.
Type ".help" for more information.
[> Object.keys(global)
[
  'global',
  'clearInterval',
  'clearTimeout',
  'setInterval',
  'setTimeout',
  'queueMicrotask',
  'clearImmediate',
  'setImmediate'
]
>
```



Onde estão as operações **require** e
os objetos **module** e **exports**?

node-v0.x-archive/node.js a... +

GitHub, Inc. (US) https://github.com/nodejs/node-v0.x-archive/blob/master/src/node.js Pesquisar

This repository Search Pull requests Issues Gist

nodejs / node-v0.x-archive Watch 2,546 Star 37,415 Fork 8,443

Code Issues 597 Pull requests 43 Wiki Pulse Graphs

Branch: master node-v0.x-archive / src / node.js Find file Copy path

misterdjules src: enable strict mode in all builtin modules ef43443 on 3 Mar 2015

63 contributors and others

819 lines (665 sloc) | 23.3 KB Raw Blame History

```
1 // Copyright Joyent, Inc. and other Node contributors.  
2 //  
3 // Permission is hereby granted, free of charge, to any person obtaining a  
4 // copy of this software and associated documentation files (the  
5 // "Software"), to deal in the Software without restriction, including  
6 // without limitation the rights to use, copy, modify, merge, publish,  
7 // distribute, sublicense, and/or sell copies of the Software, and to permit  
8 // persons to whom the Software is furnished to do so, subject to the  
9 // following conditions:  
10 //  
11 // The above copyright notice and this permission notice shall be included  
12 // in all copies or substantial portions of the Software.  
13 //  
14 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS  
15 // OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF  
16 // MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN  
17 // NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,  
18 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR  
19 // OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE  
20 // USE OR OTHER DEALINGS IN THE SOFTWARE.  
21 //  
22 // Hello, and welcome to hacking node.js!  
23 //  
24 // This file is invoked by node::Load in src/node.cc, and responsible for  
25 // bootstrapping the node.js core. Special caution is given to the performance
```

<https://github.com/nodejs/node-v0.x-archive/blob/master/src/node.js>

```
794 NativeModule.wrap = function(script) {
795   return NativeModule.wrapper[0] + script + NativeModule.wrapper[1];
796 }
797
798 NativeModule.wrapper = [
799   '(function (exports, require, module, __filename, __dirname) { ',
800   '\n});'
801 ];
802
803 NativeModule.prototype.compile = function() {
804   var source = NativeModule.getSource(this.id);
805   source = NativeModule.wrap(source);
806
807   var fn = runInThisContext(source, { filename: this.filename });
808   fn(this.exports, NativeModule.require, this, this.filename);
809
810   this.loaded = true;
811 };
812
```

The screenshot shows a Mac OS X-style interface for a code editor, likely Visual Studio Code, displaying a Node.js project. The left sidebar shows a tree view with a node named 'NODEJS' containing files: 'generator.js', 'main.js', and 'package.json'. The 'main.js' file is currently selected and open in the main editor area. The code in 'main.js' is as follows:

```
1 global.max = 100;
2 const generator = require("./generator");
3 console.log(generator.generate());
```

The terminal below shows the output of running the script:

```
rodrigobranas nodejs $ node main.js
95
rodrigobranas nodejs $
```

A screenshot of a code editor interface showing a Node.js project structure. The left sidebar displays the file tree under the 'NODEJS' category:

- generator.js
- main.js
- package.json

The main editor area has two tabs open: 'main.js' (active) and 'generator.js'. The code in 'main.js' is:

```
1 module.exports.generate = function () {
2     return Math.floor(Math.random() * max);
3 }
4 
```

A screenshot of a code editor showing a Node.js project structure. The left sidebar shows the 'EXPLORER' view with a tree titled 'NODEJS' containing files: 'generator.js', 'main.js', and 'package.json'. The main workspace has two tabs open: 'main.js' (active) and 'generator.js'. The 'main.js' tab contains the following code:

```
1 module.exports.generate = function () {
2     return Math.floor(Math.random() * global.max);
3 }
4
```

A close-up photograph of a yellow caution tape. The word "CAUTION" is printed in large, bold, black capital letters. The tape is positioned diagonally across the frame, with a blurred background of green foliage visible behind it.

CAUTION

É possível criar uma variável global
sem utilizar const, let e var

A screenshot of the Visual Studio Code (VS Code) interface. The title bar shows "main.js — nodejs". The left sidebar has "EXPLORER" and "NODEJS" sections. Under "NODEJS", "generator.js" and "main.js" are listed, with "main.js" currently selected. Below them is a package.json file. The main editor area displays the following code:

```
1 max = 100;
2 const generator = require("./generator");
3 console.log(generator.generate());
```



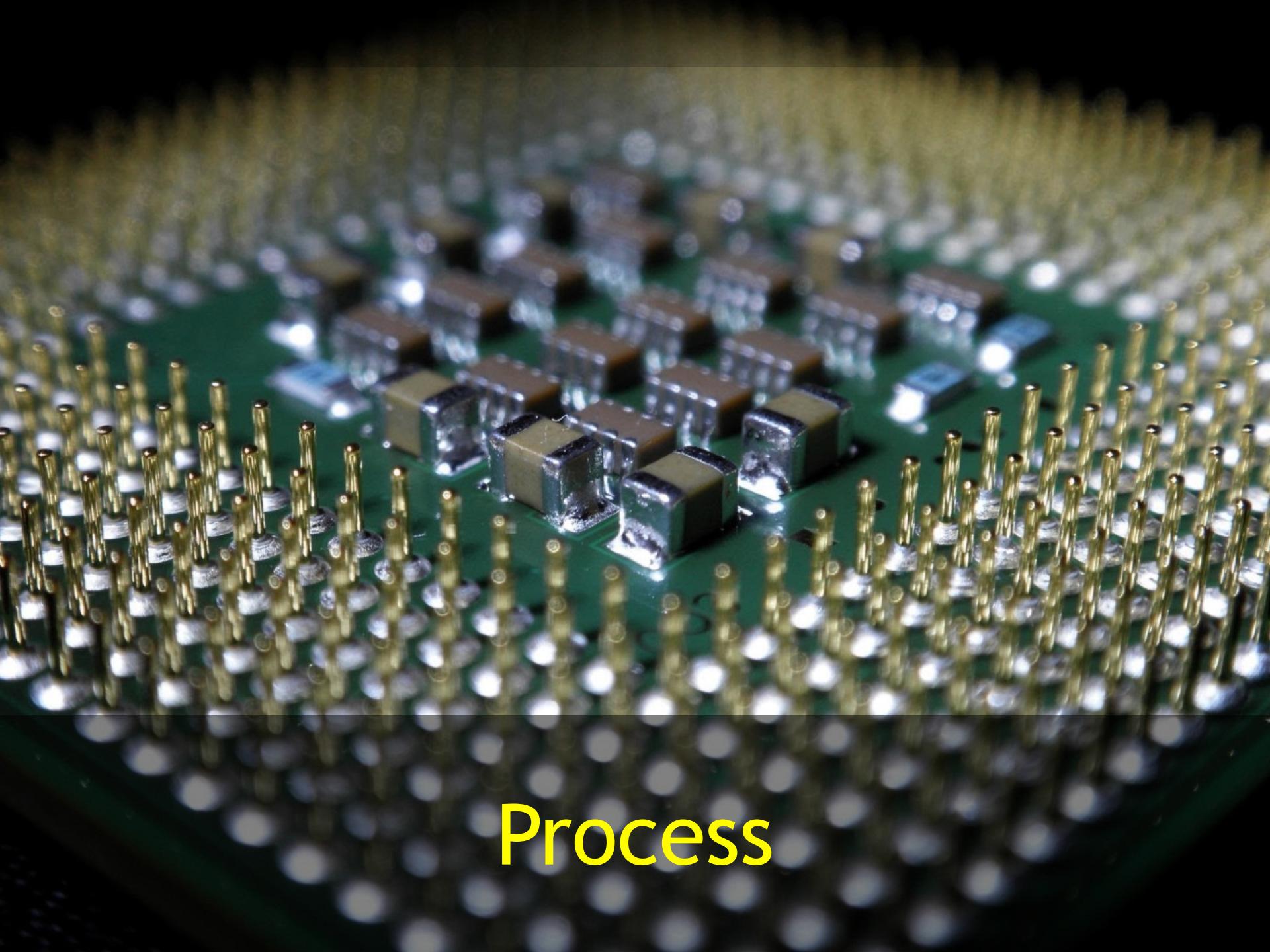
Como fazer para criar variáveis
globais **sem utilizar o escopo global?**

A screenshot of a code editor interface showing a Node.js project structure. The left sidebar, titled 'EXPLORER', displays a tree view under 'NODEJS' with files: config.js, generator.js, main.js, and package.json. The main workspace shows two tabs: 'main.js' and 'config.js'. The 'config.js' tab is active, containing the following code:

```
1 exports.max = 100;
```

A screenshot of a code editor showing a Node.js project structure. The left sidebar shows files in the 'NODEJS' folder: config.js, generator.js, main.js, and package.json. The 'generator.js' file is currently open in the main editor area, displaying the following code:

```
1 const { max } = require("./config");
2
3 module.exports.generate = function () {
4     return Math.floor(Math.random() * max);
5 }
6
```



Process

node-v0.x-archive/node.js a... +

GitHub, Inc. (US) https://github.com/nodejs/node-v0.x-archive/blob/master/src/node.js Pesquisar

This repository Search Pull requests Issues Gist

nodejs / node-v0.x-archive Watch 2,546 Star 37,415 Fork 8,443

Code Issues 597 Pull requests 43 Wiki Pulse Graphs

Branch: master node-v0.x-archive / src / node.js Find file Copy path

misterdjules src: enable strict mode in all builtin modules ef43443 on 3 Mar 2015

63 contributors and others

819 lines (665 sloc) | 23.3 KB Raw Blame History

```
1 // Copyright Joyent, Inc. and other Node contributors.  
2 //  
3 // Permission is hereby granted, free of charge, to any person obtaining a  
4 // copy of this software and associated documentation files (the  
5 // "Software"), to deal in the Software without restriction, including  
6 // without limitation the rights to use, copy, modify, merge, publish,  
7 // distribute, sublicense, and/or sell copies of the Software, and to permit  
8 // persons to whom the Software is furnished to do so, subject to the  
9 // following conditions:  
10 //  
11 // The above copyright notice and this permission notice shall be included  
12 // in all copies or substantial portions of the Software.  
13 //  
14 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS  
15 // OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF  
16 // MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN  
17 // NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,  
18 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR  
19 // OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE  
20 // USE OR OTHER DEALINGS IN THE SOFTWARE.  
21 //  
22 // Hello, and welcome to hacking node.js!  
23 //  
24 // This file is invoked by node::Load in src/node.cc, and responsible for  
25 // bootstrapping the node.js core. Special caution is given to the performance
```

<https://github.com/nodejs/node-v0.x-archive/blob/master/src/node.js>

```
1/2
173 startup.globalVariables = function() {
174     global.process = process;
175     global.global = global;
176     global.GLOBAL = global;
177     global.root = global;
178     global.Buffer = NativeModule.require('buffer').Buffer;
179     process.domain = null;
180     process._exiting = false;
181 };
182
```



nodejs — node — 81x24

```
[> Object.keys(process)
```

```
[  
  'version',  
  'versions',  
  'arch',  
  'platform',  
  'release',  
  '_rawDebug',  
  'moduleLoadList',  
  'binding',  
  '_linkedBinding',  
  '_events',  
  '_eventsCount',  
  '_maxListeners',  
  'domain',  
  '_exiting',  
  'config',  
  'dlopen',  
  'uptime',  
  '_getActiveRequests',  
  '_getActiveHandles',  
  'reallyExit',  
  '_kill',  
  'cpuUsage',
```

A screenshot of the Visual Studio Code (VS Code) interface. The left sidebar shows a file tree under the 'EXPLORER' tab, with a nodejs folder expanded. Inside, files include config.js, generator.js, main.js, package.json, and process.js, which is currently selected. The main workspace contains a code editor with the following content:

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

The bottom right corner of the code editor has a small preview window showing the current code. Below the code editor is a terminal window titled 'TERMINAL'. It displays the command 'node process.js -a 100 -b 2 -c 12' followed by the output of the program's execution:

```
rodrigobranas nodejs $ node process.js -a 100 -b 2 -c 12  
/Users/rodrigobranas/.nvm/versions/node/v15.14.0/bin/node  
/Users/rodrigobranas/development/workspace/branas/fullstackjs/nod  
ejs/process.js  
-a  
100  
-b  
2  
-c  
12  
rodrigobranas nodejs $
```

The terminal also includes standard control icons like copy, paste, and close.



Exibindo informações do processo

A screenshot of the Visual Studio Code (VS Code) interface. The title bar shows "process.js — nodejs". The left sidebar has "EXPLORER" and a "NODEJS" section containing files: config.js, generator.js, main.js, package.json, and process.js (which is selected). The main editor area shows the code for process.js:

```
1 const [option] = process.argv.slice(2);
2 switch (option) {
3     case 'a':
4         console.log('pid: ' + process.pid);
5         break;
6     case 'b':
7         console.log('title: ' + process.title);
8         break;
9     case 'c':
10        console.log('arch: ' + process.arch);
11        break;
12    case 'd':
13        console.log('platform: ' + process.platform);
14        break;
15 }
16
```

The screenshot shows a Mac OS X desktop environment with a Visual Studio Code window open. The window title is "process.js — nodejs".

The Explorer sidebar on the left shows a folder named "NODEJS" containing files: config.js, generator.js, main.js, package.json, and process.js. The "process.js" file is currently selected.

The main editor area displays the following JavaScript code:

```
1 const [option] = process.argv.slice(2);
2 switch (option) {
3     case 'a':
4         console.log('pid: ' + process.pid);
5         break;
6     case 'b':
7         console.log('title: ' + process.title);
8         break;
9     case 'c':
10        console.log('arch: ' + process.arch);
11        break;
```

Below the editor, the "TERMINAL" tab is active, showing the output of running the script with different arguments:

```
rodrigobranas nodejs $ node process.js a
pid: 51602
rodrigobranas nodejs $ node process.js b
title: node
rodrigobranas nodejs $ node process.js c
arch: arm64
rodrigobranas nodejs $ node process.js d
platform: darwin
rodrigobranas nodejs $ █
```



Standard Streams

São canais de comunicação, utilizados para realizar operações de entrada e saída, entre o programa e o ambiente onde ele está sendo executado.

Input

Output

Error

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface running on a Mac OS X system. The title bar indicates the file is named "console.js — nodejs".

The Explorer sidebar on the left shows a folder named "NODEJS" containing several files: config.js, console.js (which is currently selected), generator.js, main.js, package.json, and process.js.

The main editor area displays the contents of the "console.js" file:

```
1 const konsole = {  
2     log(msg) {  
3         |     process.stdout.write(msg + "\n");  
4     }  
5 };  
6  
7 konsole.log("Node.js");  
8
```

The terminal below the editor shows the output of running the script:

```
rodrigobranas nodejs $ node console.js  
Node.js  
rodrigobranas nodejs $ █
```

The bottom navigation bar includes tabs for PROBLEMS, OUTPUT, TERMINAL (which is underlined, indicating it is active), and other options like zsh, +, and -.

A screenshot of the Visual Studio Code (VS Code) interface. The left sidebar shows a tree view with a node named 'NODEJS' expanded, containing files: config.js, console.js (which is selected), generator.js, main.js, package.json, and process.js. The main area shows the code for 'console.js':

```
1 const konsole = {
2     log(msg) {
3         process.stdout.write(msg + "\n");
4     },
5     error(msg) {
6         process.stderr.write(msg + "\n");
7     }
8 };
9
10 konsole.error("Node.js");
11
```

The status bar at the bottom indicates the current workspace is 'nodejs'. Below the editor, there are tabs for PROBLEMS, OUTPUT, TERMINAL, and other options. The TERMINAL tab is active, showing the command-line output:

```
rodrigobranas nodejs $ node console.js
Node.js
rodrigobranas nodejs $ █
```

The bottom left corner of the interface has a 'OUTLINE' button.



Qual é a diferença entre **output** e **error**?

Eles podem ser **tratados** e
redirecionados de formas diferentes



The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. On the left, the Explorer sidebar displays a project structure under the 'NODEJS' folder, including files like config.js, console.js, error.log, generator.js, main.js, out.log, package.json, and process.js. The 'console.js' file is currently selected and open in the central code editor. The code in 'console.js' is as follows:

```
2 log(msg) {  
3     process.stdout.write(msg + "\n");  
4 },  
5 error(msg) {  
6     process.stderr.write(msg + "\n");  
7 }  
8 };  
9  
10 konsole.log("Node.js: log");  
11 konsole.error("Node.js: error");  
12
```

Below the editor, the 'TERMINAL' tab is active, showing the command-line output of running the script:

```
rodrigobranas nodejs $ node console.js 1> out.log 2> error.log  
rodrigobranas nodejs $ █
```

The bottom left corner of the interface shows the 'OUTLINE' icon.

A screenshot of the Visual Studio Code (VS Code) interface. The window title is "out.log — nodejs".

The Explorer sidebar on the left shows a project structure under the "NODEJS" folder:

- config.js
- console.js
- error.log
- generator.js
- main.js
- out.log (selected)
- package.json
- process.js

The "TERMINAL" tab is active at the bottom, showing the command-line output:

```
rodrigobranas nodejs $ node console.js 1> out.log 2> error.log
rodrigobranas nodejs $
```

The "out.log" tab in the center shows the content of the log file:

```
1 Node.js: log
2
```

A screenshot of the Visual Studio Code (VS Code) interface. The window title is "error.log — nodejs". The left sidebar shows a project structure under "NODEJS" with files: config.js, console.js, generator.js, main.js, out.log, package.json, and process.js. The file "error.log" is selected in the sidebar. The main area has tabs for "console.js" and "error.log", with "error.log" currently active. The content of the "error.log" tab shows two lines of text: "1 Node.js: error" and "2". Below this is a terminal window with tabs for "PROBLEMS", "OUTPUT", and "TERMINAL", with "TERMINAL" selected. The terminal output shows the command "rodrigobranas nodejs \$ node console.js 1> out.log 2> error.log" followed by a prompt "rodrigobranas nodejs \$". The bottom left corner of the interface shows the "OUTLINE" icon.

```
1 Node.js: error
2
```

```
rodrigobranas nodejs $ node console.js 1> out.log 2> error.log
rodrigobranas nodejs $
```



Timers



Por meio dos **timers**, é possível agendar a execução de funções

The screenshot shows a VS Code interface with the following details:

- EXPLORER** sidebar: Shows a folder named "NODEJS" containing "main.js" and "package.json".
- main.js** editor tab: Displays the following code:

```
1 console.log("1", new Date());
2 setTimeout(function () {
3     console.log("2", new Date());
4 }, 2000);
5
```
- TERMINAL** tab: Shows the command "node main" being run, followed by the output:

```
rodrigobranas nodejs $ node main
1 2022-03-31T19:51:55.769Z
2 2022-03-31T19:51:57.779Z
rodrigobranas nodejs $ █
```

Criando um timer com setTimeout



É necessário importar algum módulo
para utilizar a função **setTimeout**?

node-v0.x-archive/node.js a... +

GitHub, Inc. (US) https://github.com/nodejs/node-v0.x-archive/blob/master/src/node.js Pesquisar

This repository Search Pull requests Issues Gist

nodejs / node-v0.x-archive Watch 2,546 Star 37,415 Fork 8,443

Code Issues 597 Pull requests 43 Wiki Pulse Graphs

Branch: master node-v0.x-archive / src / node.js Find file Copy path

misterdjules src: enable strict mode in all builtin modules ef43443 on 3 Mar 2015

63 contributors and others

819 lines (665 sloc) | 23.3 KB Raw Blame History

```
1 // Copyright Joyent, Inc. and other Node contributors.  
2 //  
3 // Permission is hereby granted, free of charge, to any person obtaining a  
4 // copy of this software and associated documentation files (the  
5 // "Software"), to deal in the Software without restriction, including  
6 // without limitation the rights to use, copy, modify, merge, publish,  
7 // distribute, sublicense, and/or sell copies of the Software, and to permit  
8 // persons to whom the Software is furnished to do so, subject to the  
9 // following conditions:  
10 //  
11 // The above copyright notice and this permission notice shall be included  
12 // in all copies or substantial portions of the Software.  
13 //  
14 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS  
15 // OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF  
16 // MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN  
17 // NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,  
18 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR  
19 // OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE  
20 // USE OR OTHER DEALINGS IN THE SOFTWARE.  
21 //  
22 // Hello, and welcome to hacking node.js!  
23 //  
24 // This file is invoked by node::Load in src/node.cc, and responsible for  
25 // bootstrapping the node.js core. Special caution is given to the performance
```

<https://github.com/nodejs/node-v0.x-archive/blob/master/src/node.js>

```
183 startup.globalTimeouts = function() {
184   global.setTimeout = function() {
185     var t = NativeModule.require('timers');
186     return t.setTimeout.apply(this, arguments);
187   };
188
189   global.setInterval = function() {
190     var t = NativeModule.require('timers');
191     return t.setInterval.apply(this, arguments);
192   };
193
194   global.clearTimeout = function() {
195     var t = NativeModule.require('timers');
196     return t.clearTimeout.apply(this, arguments);
197   };
198
199   global.clearInterval = function() {
200     var t = NativeModule.require('timers');
201     return t.clearInterval.apply(this, arguments);
202   };
203
204   global.setImmediate = function() {
205     var t = NativeModule.require('timers');
206     return t.setImmediate.apply(this, arguments);
207   };
208
209   global.clearImmediate = function() {
210     var t = NativeModule.require('timers');
211     return t.clearImmediate.apply(this, arguments);
212   };
213 }
```

A screenshot of the Visual Studio Code (VS Code) interface. The left sidebar shows a project structure under the 'NODEJS' folder, containing 'main.js' and 'package.json'. The main editor tab is titled 'main.js — nodejs' and contains the following code:

```
1 setTimeout(function () {
2   console.log("2", new Date());
3 }, 2000);
4 console.log("1", new Date());
5
```

The bottom right corner of the editor has a small preview window showing the current code. Below the editor, the terminal tab is active, showing the output of running the script:

```
rodrigobranas nodejs $ node main
1 2022-03-31T19:52:35.091Z
2 2022-03-31T19:52:37.097Z
rodrigobranas nodejs $ █
```

Invertendo a ordem de execução



Por que ao inverter a ordem
o resultado é o mesmo?

Para evitar o bloqueio do Event Loop, que é executado em uma única thread, as funções invocadas pelos timers ficam aguardando na Event Queue

The screenshot shows a VS Code interface with the following details:

- EXPLORER** sidebar: Shows a tree view with a node labeled **NODEJS** expanded, containing files **main.js** and **package.json**.
- main.js** editor tab: The active file contains the following JavaScript code:

```
1 var a = setTimeout(function () {
2   console.log("1", new Date());
3 }, 3000);
4 var b = setTimeout(function () {
5   console.log("2", new Date());
6 }, 3000);
7 clearTimeout(a);
8
```
- TERMINAL** tab: The terminal window shows the command `node main.js` being run, followed by the output:

```
rodrigobranas nodejs $ node main.js
2 2022-03-31T20:36:57.021Z
rodrigobranas nodejs $
```

Cancelando um timer com clearTimeout clearTimeout.js



É possível também criar um timer que execute a cada período de tempo

The screenshot shows a VS Code interface on a Mac OS X system. The left sidebar has 'EXPLORER' and 'NODEJS' sections, with 'main.js' selected. The main editor tab is 'main.js'. The code in main.js is:

```
1 var interval = function (callback, time) {  
2     setTimeout(function() {  
3         callback();  
4         interval(callback, time);  
5     }, time);  
6 };  
7 interval(function() {  
8     console.log("R", new Date());  
9 }, 1000);  
10
```

The terminal tab is active, showing the output of running the script:

```
rodrigobranas nodejs $ node main.js  
R 2022-03-31T20:37:52.259Z  
R 2022-03-31T20:37:53.268Z  
R 2022-03-31T20:37:54.273Z  
R 2022-03-31T20:37:55.279Z
```

Criando um timer com setTimeout



Não tem nada mais simples e direto?

The screenshot shows a Mac OS X desktop environment with a dark theme. A Visual Studio Code window is open, displaying a Node.js project. The Explorer sidebar on the left shows two files: `main.js` and `package.json`. The `main.js` file content is:

```
1  setInterval(function () {
2      console.log("R", new Date());
3  }, 1000);
4
```

The terminal tab at the bottom shows the output of running `node main.js`:

```
rodrigobranas nodejs $ node main.js
R 2022-03-31T20:38:52.965Z
R 2022-03-31T20:38:53.971Z
R 2022-03-31T20:38:54.975Z
R 2022-03-31T20:38:55.980Z
```

Criando um timer com setInterval

A screenshot of the Visual Studio Code (VS Code) interface. The left sidebar shows a file tree with a 'NODEJS' folder containing 'main.js' and 'package.json'. The main editor window displays the code for 'main.js':

```
1 const i = setInterval(function () {
2   console.log("R", new Date());
3 }, 1000);
4 setTimeout(function () {
5   clearInterval(i);
6 }, 5000);
7
```

The bottom right corner of the editor has a small preview window showing the output of the script. The terminal tab at the bottom shows the command-line output:

```
rodrigobranas nodejs $ node main.js
R 2022-03-31T20:39:58.913Z
R 2022-03-31T20:39:59.918Z
R 2022-03-31T20:40:00.923Z
R 2022-03-31T20:40:01.928Z
rodrigobranas nodejs $ █
```

Cancelando um timer com clearInterval



E se a intenção for apenas colocar a
função na fila, para executar depois?

The screenshot shows a Mac OS X desktop environment with a dark theme. A Visual Studio Code window is open, displaying a Node.js project named 'nodejs'. The 'EXPLORER' sidebar on the left shows files 'main.js' and 'package.json' under a folder named 'NODEJS'. The 'main.js' file content is:

```
1 console.log("1", new Date());
2 setTimeout(function() {
3   console.log("2", new Date());
4 }, 0);
5 console.log("3", new Date());
6
```

The 'TERMINAL' tab at the bottom shows the command 'node main.js' being run, and the output is:

```
rodrigobranas nodejs $ node main.js
1 2022-03-31T20:41:05.813Z
3 2022-03-31T20:41:05.818Z
2 2022-03-31T20:41:05.818Z
rodrigobranas nodejs $ █
```

Criando um timer imediato



Não tem nada mais simples e direto?

The screenshot shows a VS Code interface with the following details:

- EXPLORER** sidebar: Shows a project named "NODEJS" containing files "main.js" and "package.json".
- main.js** editor tab: Displays the following JavaScript code:

```
1 console.log("1", new Date());
2 setImmediate(function() {
3   console.log("2", new Date());
4 });
5 console.log("3", new Date());
6
```
- TERMINAL** tab: Shows the output of running the script with the command `node main.js`. The output is:

```
rodrigobranas nodejs $ node main.js
1 2022-03-31T20:42:11.830Z
3 2022-03-31T20:42:11.834Z
2 2022-03-31T20:42:11.834Z
rodrigobranas nodejs $ █
```

Criando um timer imediato



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays a 'NODEJS' folder containing 'main.js' and 'package.json'. The main editor tab is titled 'main.js' and contains the following code:

```
1 console.time('T');
2 setTimeout(function () {
3     console.timeEnd('T');
4 }, 0);
5
```

The terminal tab at the bottom shows the output of running the script:

```
rodrigobranas nodejs $ node main.js
T: 1.967ms
rodrigobranas nodejs $ █
```

Medindo a performance do setTimeout

The screenshot shows a Mac OS X desktop environment with a window for the Visual Studio Code editor. The title bar says "main.js — nodejs".

The Explorer sidebar on the left shows a folder named "NODEJS" containing "main.js" and "package.json".

The main editor area displays the following code in "main.js":

```
1 console.time('I');
2 setImmediate(function () {
3   console.timeEnd('I');
4 });
5
```

The bottom right corner of the editor has a small preview of the terminal output.

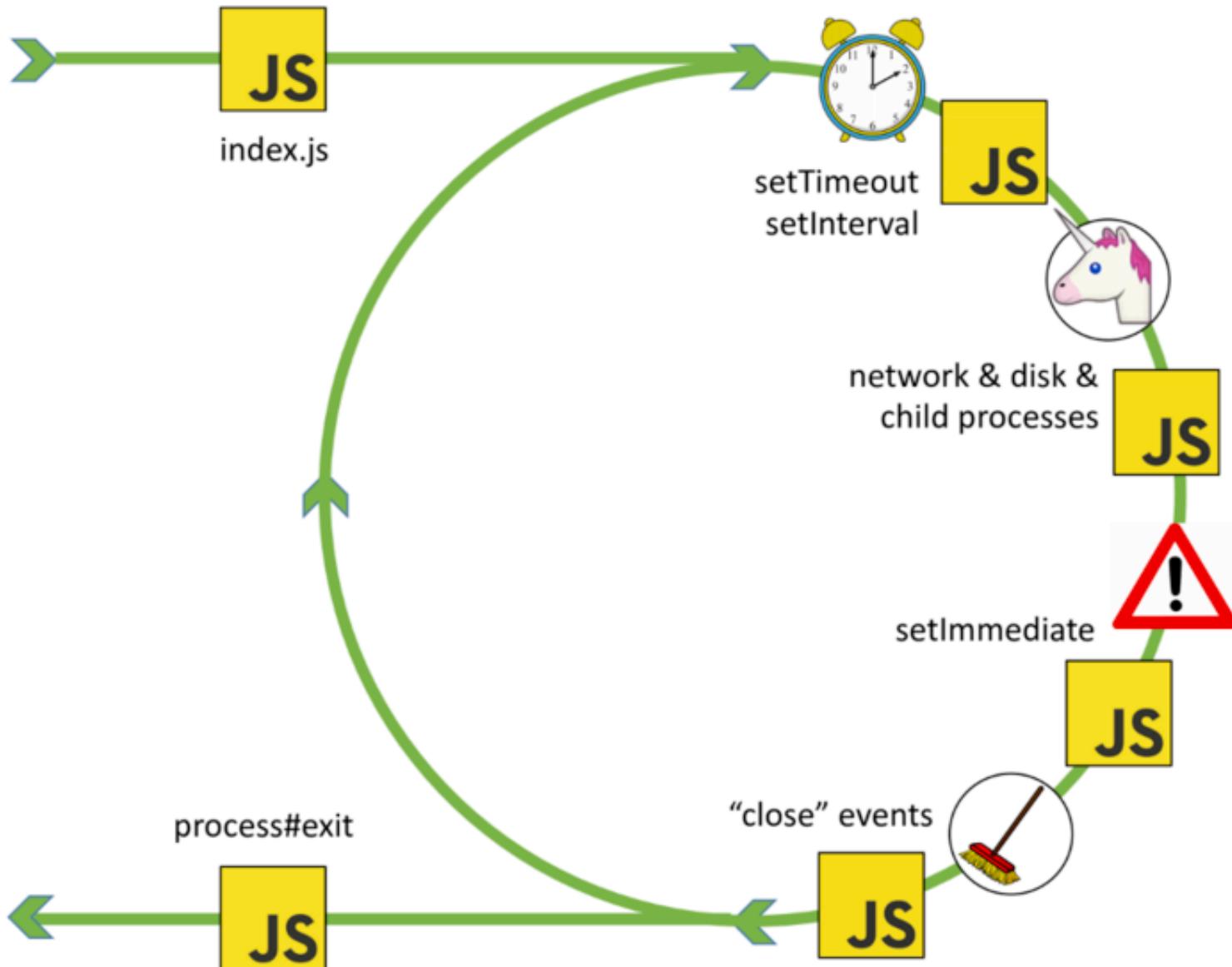
The terminal tab is active at the bottom, showing the command "node main.js" and its output:

```
rodrigobranas nodejs $ node main.js
I: 0.531ms
rodrigobranas nodejs $ █
```

Medindo a performance do setImmediate



Como os timers são executados?



EVENT LOOP



START



CALLBACK
QUEUES

Expired timer callbacks



EVENT LOOP



START



CALLBACK
QUEUES

Expired timer callbacks

c c

```
setTimeout(() => {  
  console.log('Timer expired!');  
});
```

EVENT LOOP



START



Expired timer callbacks

CALLBACK
QUEUES

c c

I/O polling and callbacks

c c c c

EVENT LOOP



START



Expired timer callbacks

CALLBACK
QUEUES

c c



I/O polling and callbacks

c c c c

```
fs.readFile('file.txt', (e, d) => {  
  |   console.log('File read!');  
});
```

EVENT LOOP



START



Expired timer callbacks

CALLBACK
QUEUES

c c

I/O polling and callbacks

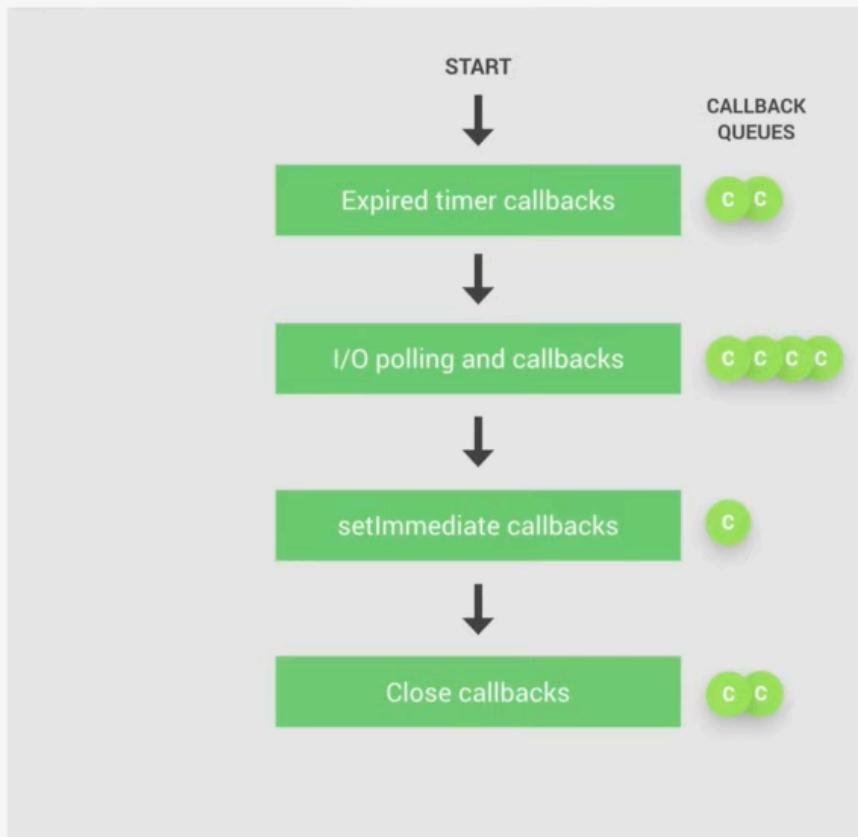
c c c c



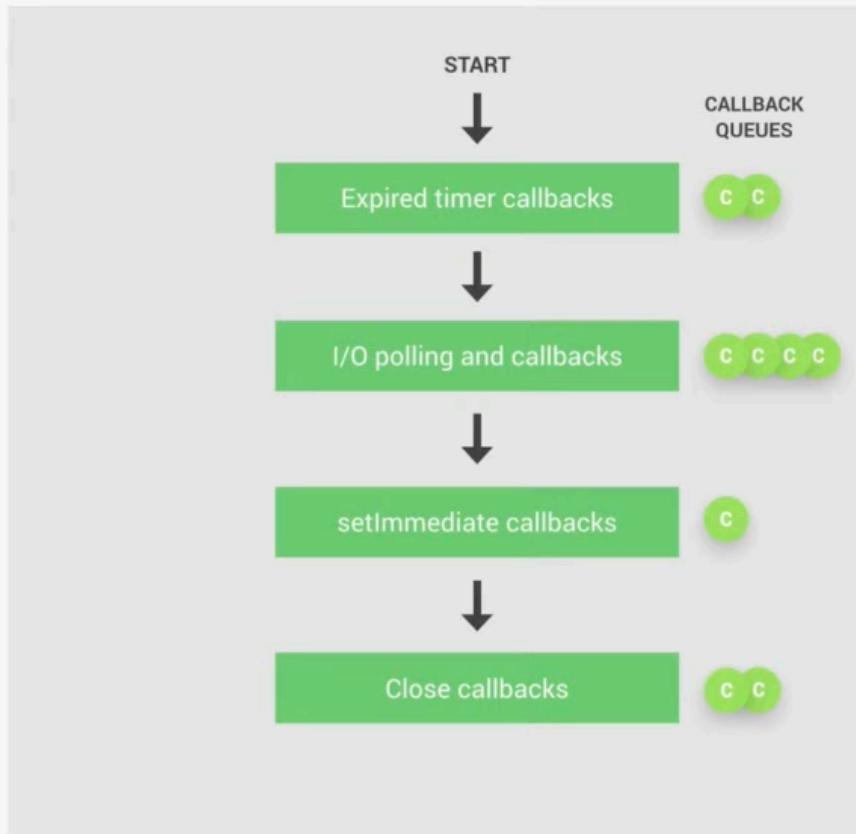
setImmediate callbacks

c

EVENT LOOP



EVENT LOOP



CALLBACK QUEUES

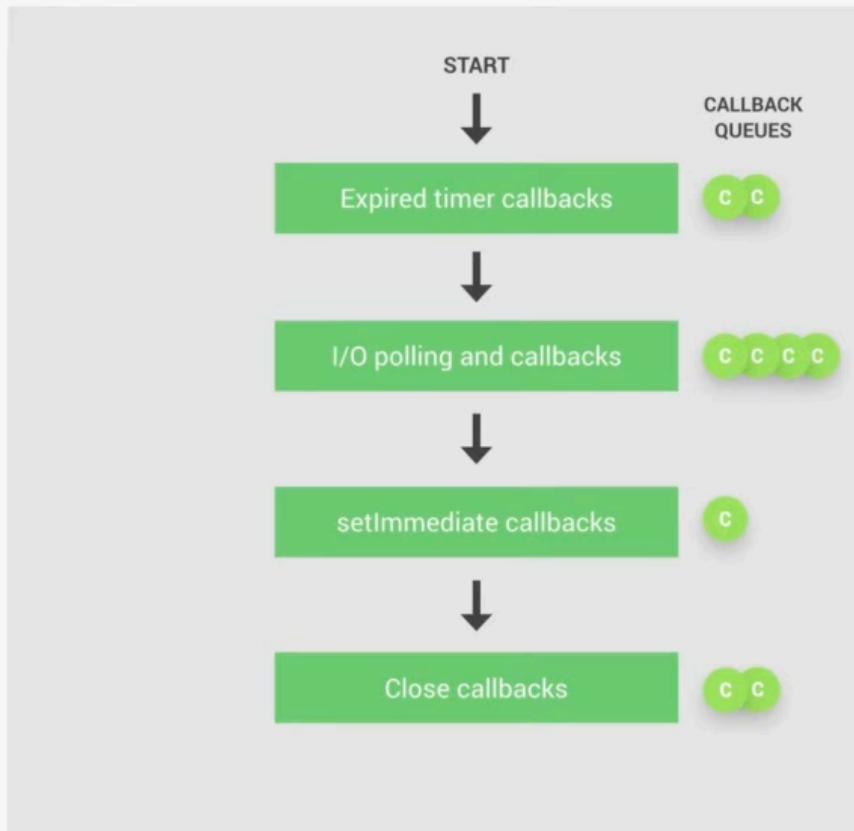


PROCESS.NEXTTICK() QUEUE



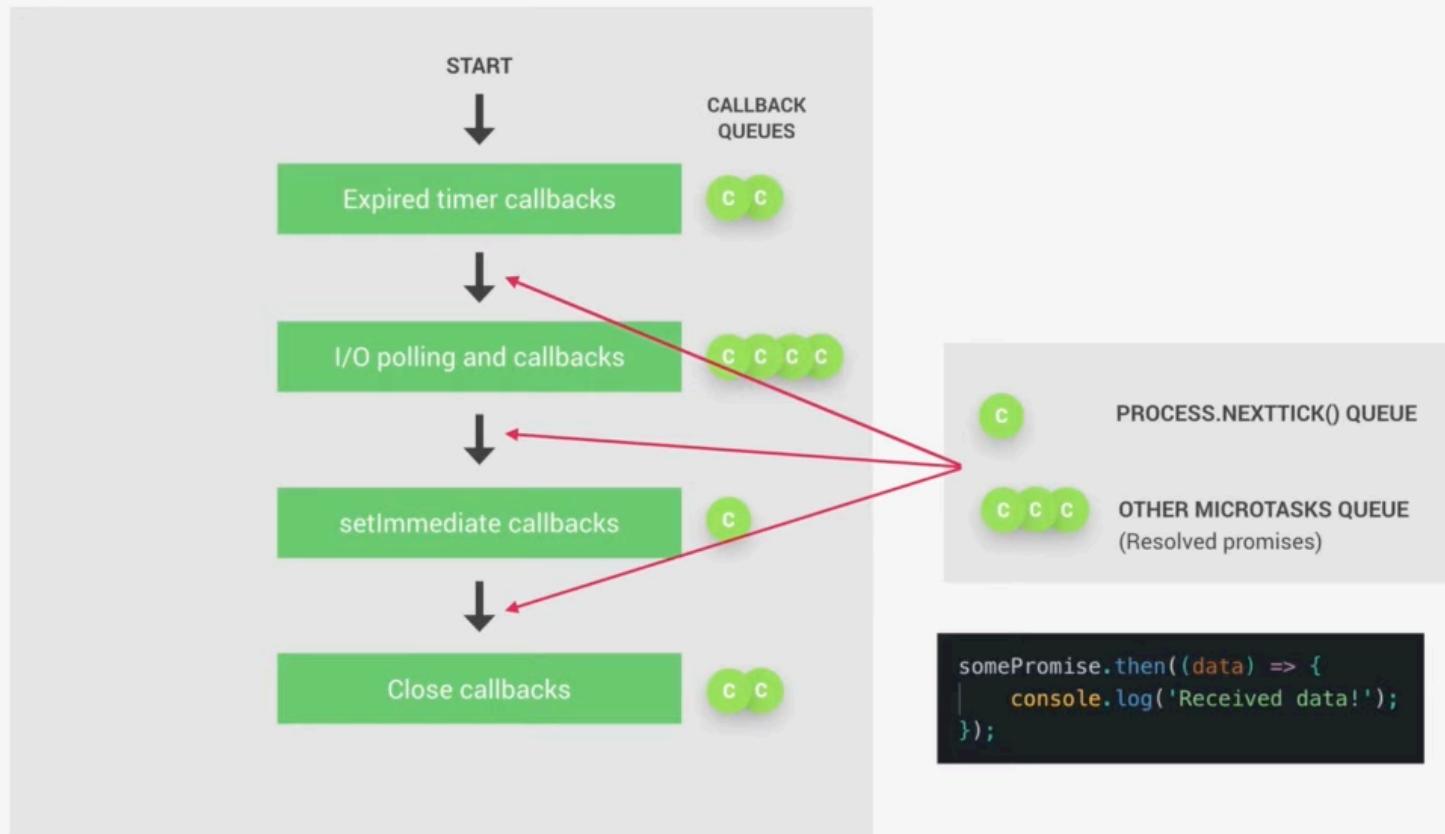
OTHER MICROTASKS QUEUE
(Resolved promises)

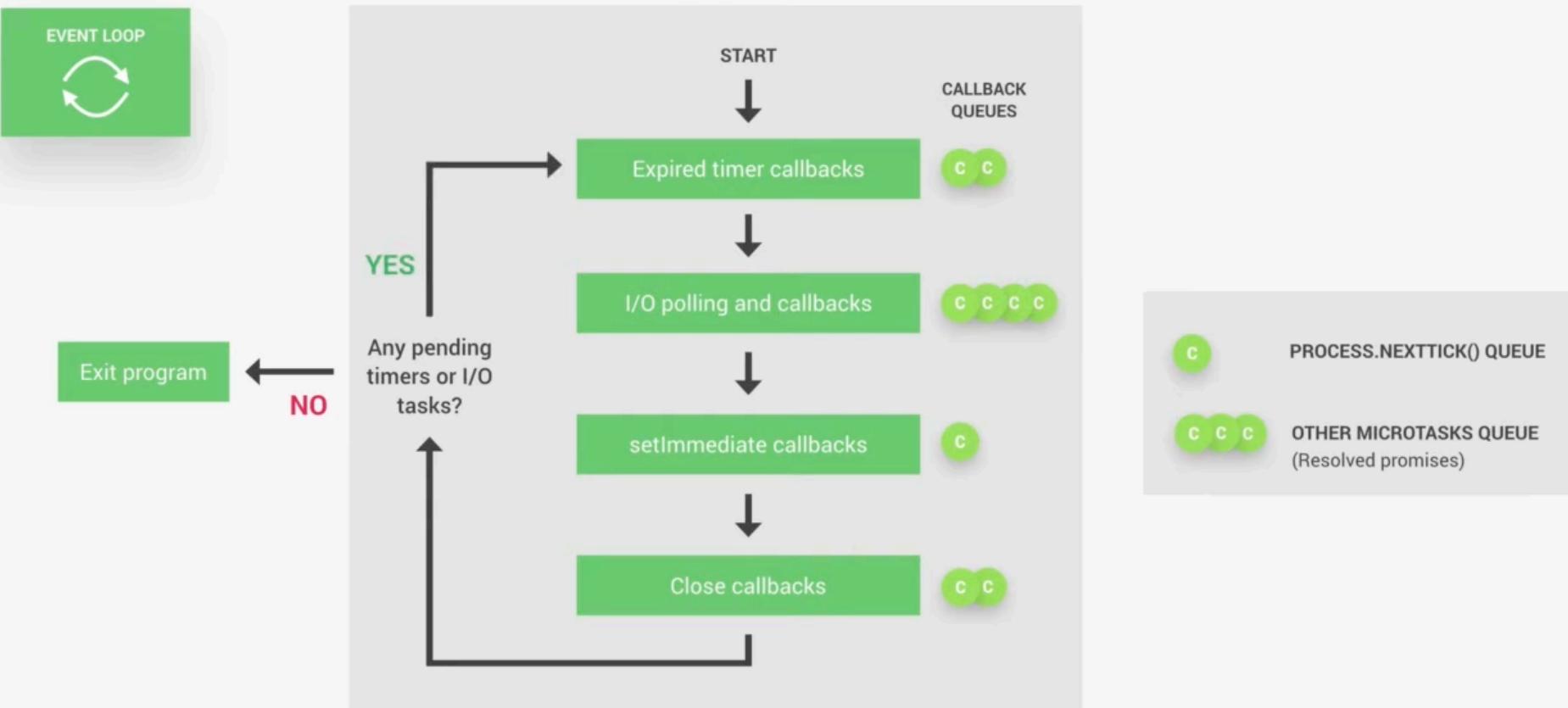
EVENT LOOP



```
somePromise.then((data) => {  
  |  console.log('Received data!');  
});
```

EVENT LOOP







Qual é a diferença entre o `nextTick` e
o `setImmediate`?