Tecnologie e applicazioni web

Node.js

Filippo Bergamasco (filippo.bergamasco@unive.it)
http://www.dais.unive.it/~bergamasco/
DAIS - Università Ca'Foscari di Venezia
Anno accademico: 2021/2022

Server-side JavaScript

Historically, creating the server part of a web application is more complex and tedious because it requires an in-depth knowledge of:

- Multi-threaded programming
- Scalability
- Security
- Server deployment
- .. etc ...

Server-side JavaScript

JavaScript was not meant to be used in a server-side environment:

- Basic memory management
- No built-in interface with the operating system
- Slow execution speed

Before Google V8 JavaScript was primarily used inside the web browser.

Node.js

Node.js is a JavaScript **runtime** composed by the Google V8 engine and an integration **layer** with the operating system providing a full-featured JavaScript environment outside the web browser.

- Open-source
- Cross-platform
- Designed to develop server-side networking applications

Why Node.js?

- Lightweight compared to traditional environments like Java, PHP, etc.
- Easy to configure and install
- Vast selection of modules (libraries) freely available and easily installable through **npm**
- Modules to connect to relational and NoSQL databases.
- JavaScript can be the only language needed for the entire web application (both server- and client-side).

Installation

Node is avaliable for Windows, OSX e Linux.
http://nodejs.org

In Linux can usually be installed with the package manager of your preferred distribution.

In OSX homebrew can be used

Batteries included

Node.js is distributed together with:

- A REPL (Read, Evaluate, Print, Loop) frontend
- A command line executable to run standalone JavaScript files
- A package manager called npm (Node Package Manager)
  - Manages the installation of additional modules
  - Resolves the dependencies among modules
  - Installs and manages additional command-line tools (like TypeScript, etc)

Node REPL

```
$ node --version
v12.14.1
$ node
> var a="filippo"
undefined
> console.log(a)
filippo
undefined
```

Code and Modules

You can simply execute a JavaScript file by running the node process:

```
$ node javascriptfile.js
```

A project is usually composed by multiple JavaScrpt files. The entry point is defined in the **package.json** file, together with some metadata about the project version, its dependencies, etc

package.json

https://docs.npmjs.com/files/package.json

```
{
      “name”: “my-project-name”,
      “main”: “main-project-js-file”,
      “version”: “0.0.1”,
      “dependencies”: {
            “colors”: “0.5.0”
      },
      “private”: true
}
```

Modules

A Node package, defined in package.json, defines a **module** that can be executed (if the main property is present) or used by other modules as a library.

npm can manage module dependences, simplifying the installation of required external modules

Modules

Modules follow the CommonJS convention:

- The require() functions allows the module loading of local and global modules
- All the variables defined inside a module are local for that module (ie. not visible to other modules)
- To export a variable it must be added in the module.exports object (or just exports since module is the global object)

npm

To install all package dependences (defined in package.json) we can use the command:

```
$ npm install
```

All the dependences will be downloaded and installed in a new directory named `node_modules` (in the same path of package.json file)

npm

Our own module can be published to the npm registry (if not marked as private) with the command:

```
$ npm publish
```

It is also possible to search for a certain module with the command

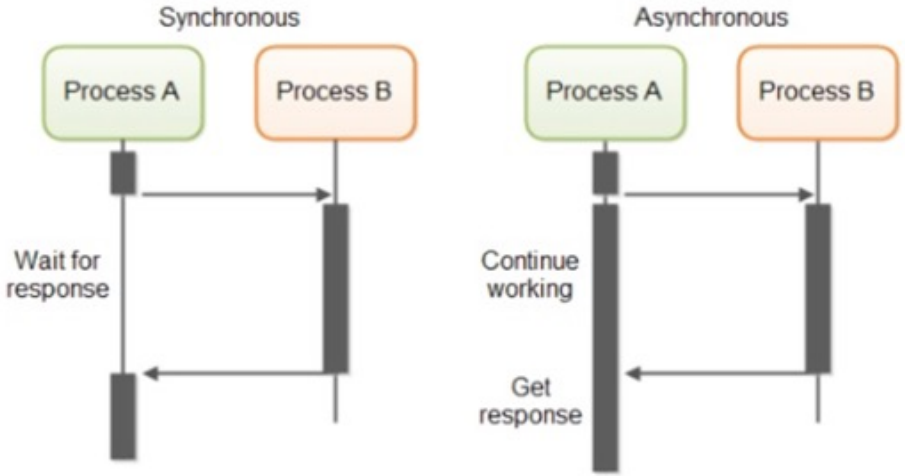
```
$ npm search <module name>
```

Node.js key concepts

1. Asynchronous (non-blocking) operations
2. Single threaded code
3. Shared-state concurrency

Async operations

The vast majority of API functions are non-blocking, executing their task in an asynchronous way

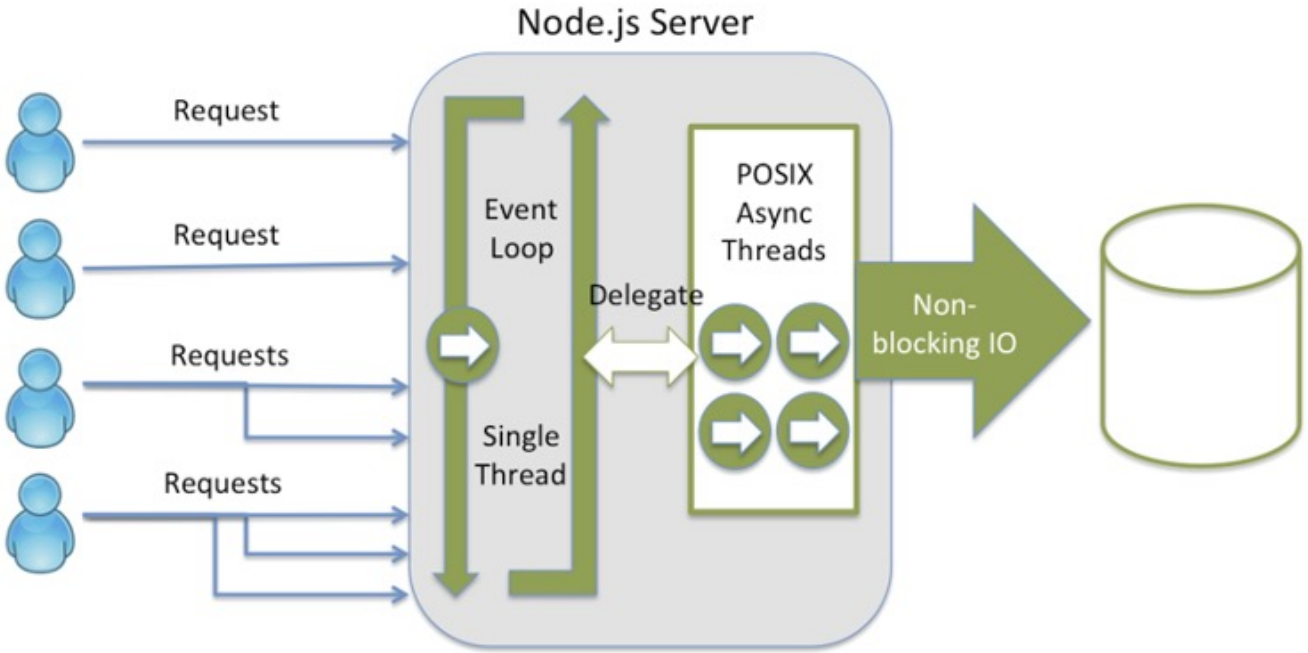


Single threaded

Node.js executes code in an asynchronous way but **not in parallel**. There is one single event-loop (like in the browser) executing all the callbacks.

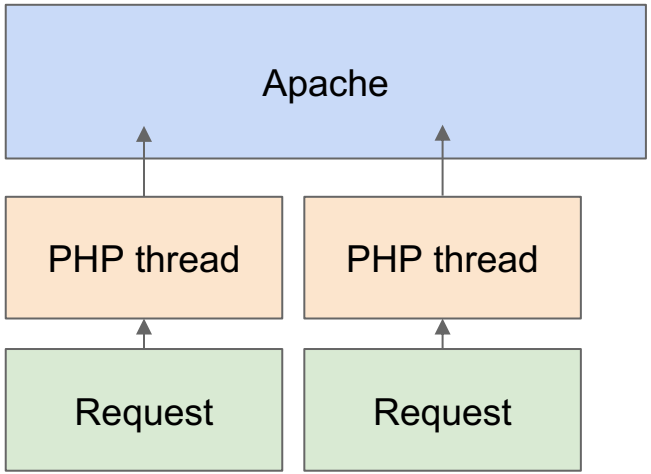
Internally, Nodejs manages a thread-pool and non-blocking IO

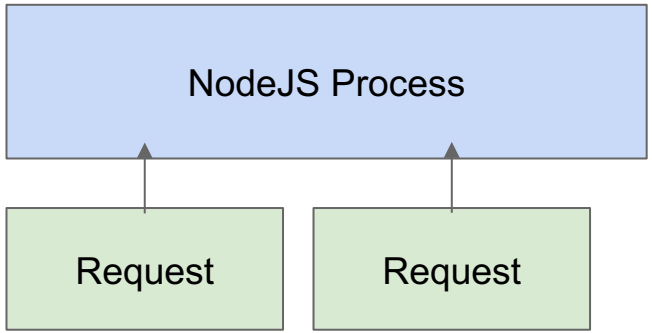
Single threaded



Shared-state concurrency

All concurrent functions observe the same consistent memory state (There is no overhead due to the creation and destruction of threads)





Use cases

When Node.js is convenient?

When the program is IO-bound. In other words, when time is mostly spent in managing the IO (disk, network, database, etc) instead of executing computationally expensive tasks

Typical of Web servers, databases, etc!

Use cases

When Node.js is NOT convenient?

For its single-threaded nature, Node.js is not efficient to execute CPU-bound programs:

- if the server is asked to manage complex CPU intensive tasks (like video encoding)
- If the operations are naturally parallelizable in a multi-threaded environment

The global object

Useful properties of the global object:

- __filename, __dirname
- setImmediate(callback[, ...args])
- setInterval(callback, delay[, ...args])
- setTimeout(callback, delay[, ...args])
- require()

APIs

Node.js provides a useful set of APIs to interface with the underlying operating system:

- Filesystem access
- TCP/UDP socket creation
- Execution of other tasks or processes
- HTTP and HTTPs built-in webserver

https://nodejs.org/api/index.html