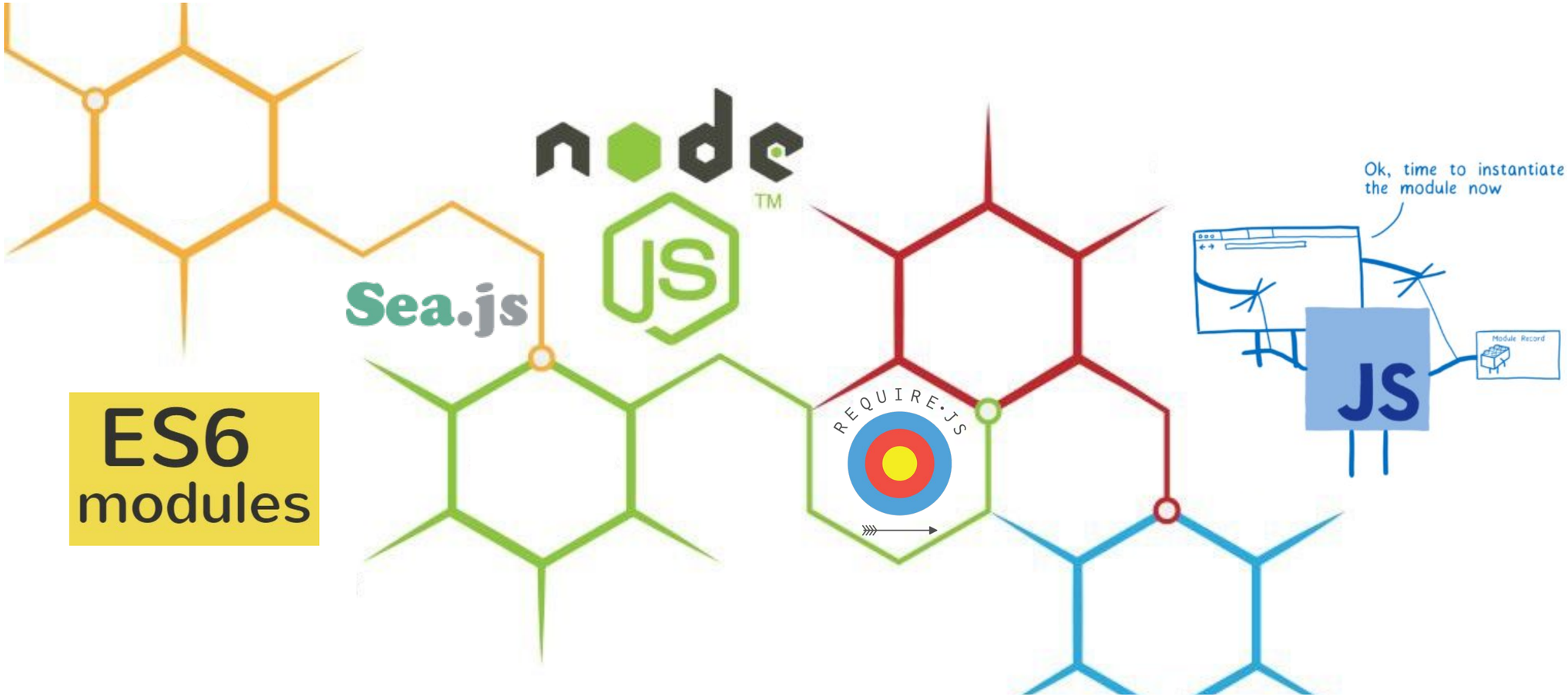
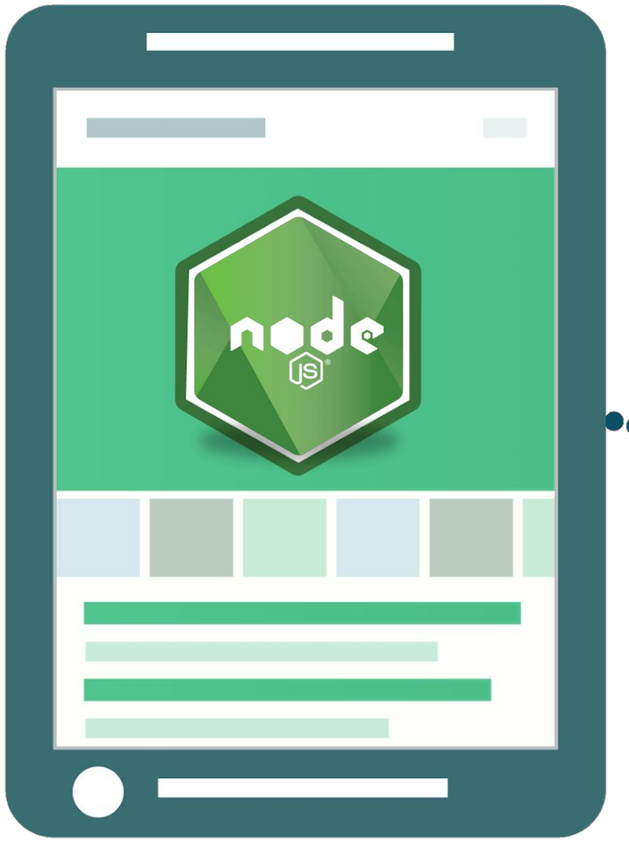


前端科普系列——

CommonJS：不是前端却革命了前端



HTTP

File System

Operate System

Cluster

Features

Child Process

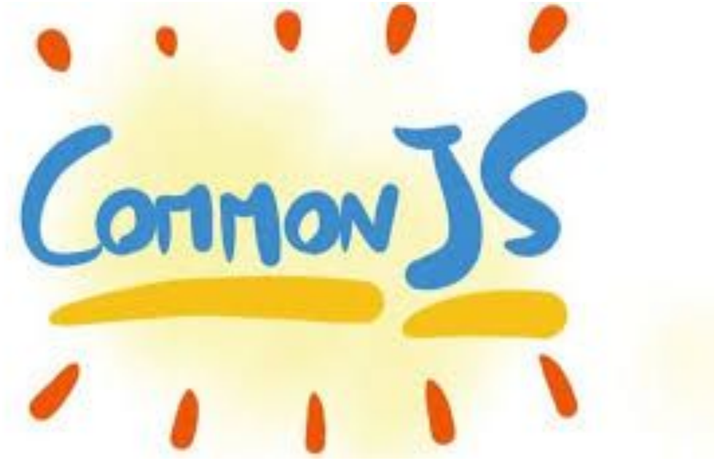
Events

Path

Net

NO? Modules

NO WAY!



What`s CommonJS ?

CommonJS was a project with the goal to establish conventions on module ecosystem for JavaScript outside of the web browser.

The primary reason for its creation was a major lack of commonly accepted form of JavaScript scripts module units which could be reusable in environments different from that provided by a conventional web browser

original

```
// index.html
<script>
var name = 'morrain'
var age = 18
</script>
```

next

```
// index.html
<script src="./mine.js"></script>

// mine.js
var name = 'morrain'
var age = 18
```

finally

```
// index.html
<script src="./mine.js"></script>
<script src="./a.js"></script>
<script src="./b.js"></script>
// mine.js
var name = 'morrain'
var age = 18
// a.js
var name = 'lilei'
var age = 15
// b.js
var name = 'hanmeimei'
var age = 13
```

Global namespace pollution
is starting to be a developer's nightmare!

```
// index.html
<script src="./mine.js"></script>
<script src="./a.js"></script>
<script src="./b.js"></script>
// mine.js
var name = 'morrain'
var age = 18
// a.js
var name = 'lilei'
var age = 15
// b.js
var name = 'hanmeimei'
var age = 13
```

How to Protect Private Variables ?

Solve global namespace pollution

```
// index.html
<script src="./mine.js"></script>
<script src="./a.js"></script>
<script src="./b.js"></script>
// mine.js
app.mine = {}
app.mine.name = 'morrain'
app.mine.age = 18
// a.js
app.moduleA = {}
app.moduleA.name = 'lilei'
app.moduleA.age = 15
// b.js
app.moduleB = {}
app.moduleB.name = 'hanmeimei'
app.moduleB.age = 13
```

```
// index.html
<script src="./mine.js"></script>
<script src="./a.js"></script>
<script src="./b.js"></script>
// mine.js
app.mine = (function(){
    var name = 'morrain'
    var age = 18
    return {
        getName: function(){
            return name
        }
    }
})()
// a.js
app.moduleA = (function(){
    var name = 'lilei'
    var age = 15
    return {
        getName: function(){
            return name
        }
    }
})()
```

The front-end needs to be modularized, and modularization not only deals with the problem of global namespace pollution and data protection, but also well solves the maintenance of dependencies between modules

The overview of CommonJS

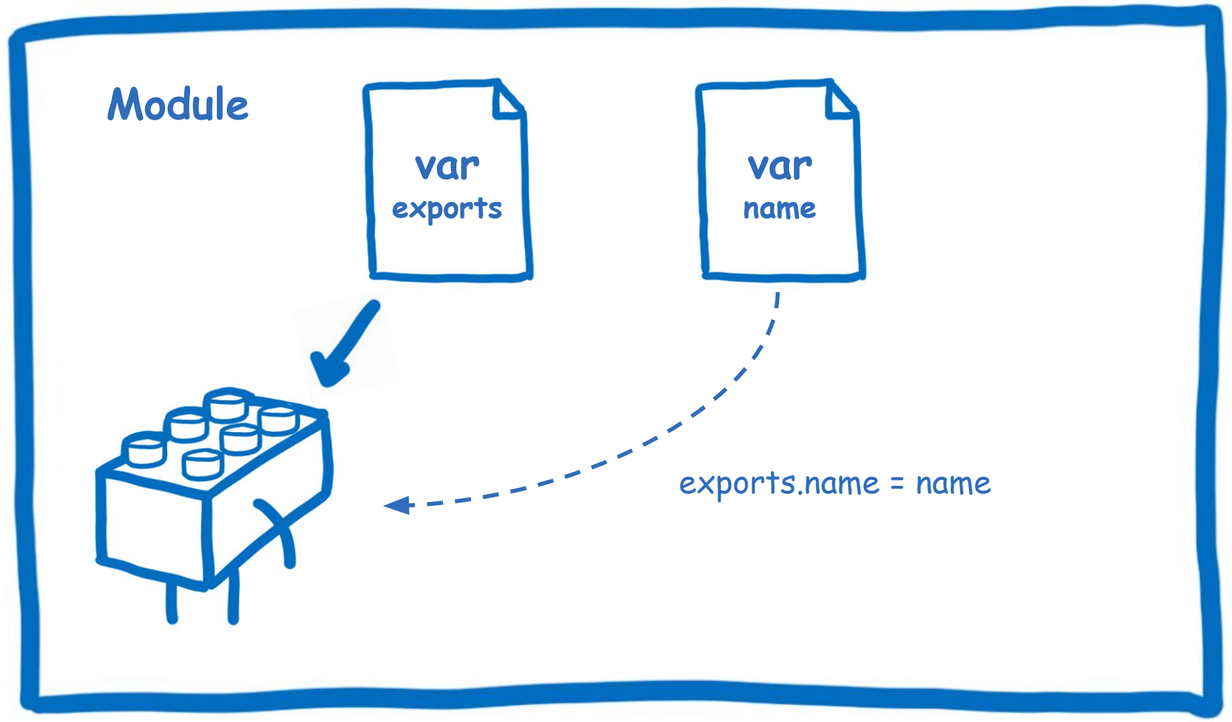
- Each file is a module
- Each file has its own scope
- Each module has `require` and `module`
- `module.exports` saves the value to export

```
// a.js
var name = 'morrain'
var age = 18
module.exports.name = name
module.exports.getAge = function(){
    return age
}

//b.js
var a = require('a.js')
console.log(a.name) // 'morrain'
console.log(a.getAge())// 18
```

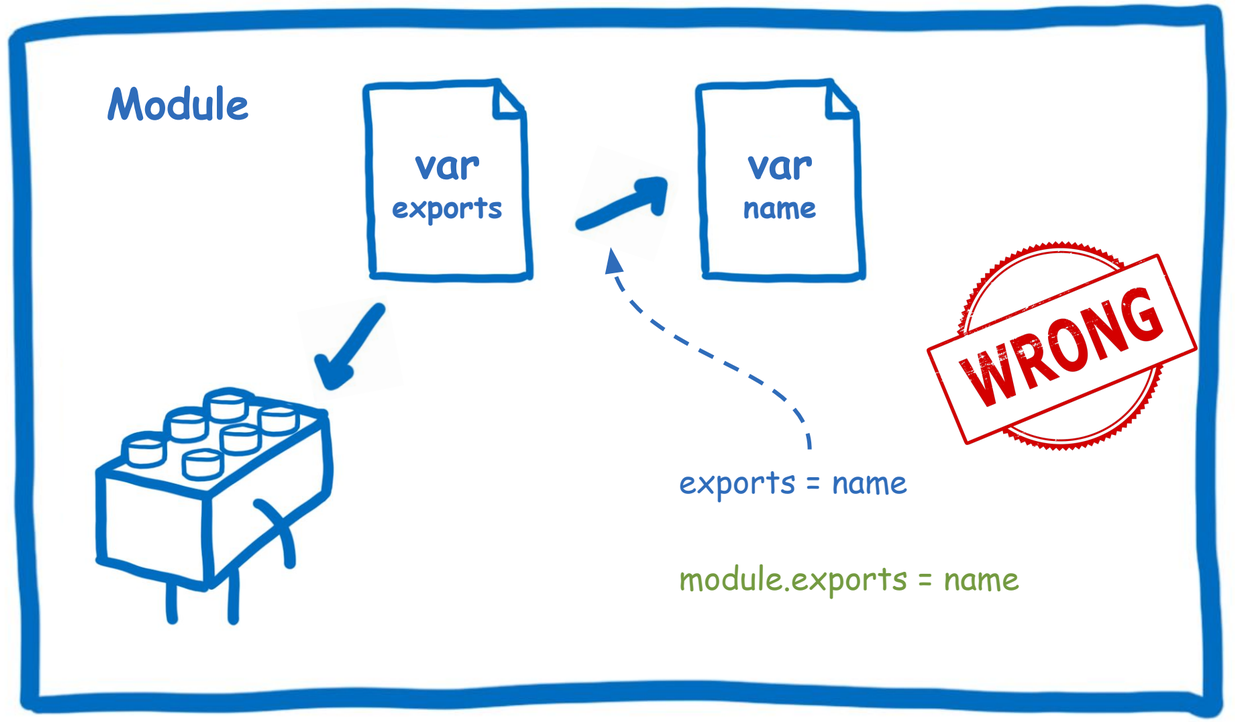
exports

var exports = module.exports

module.exports

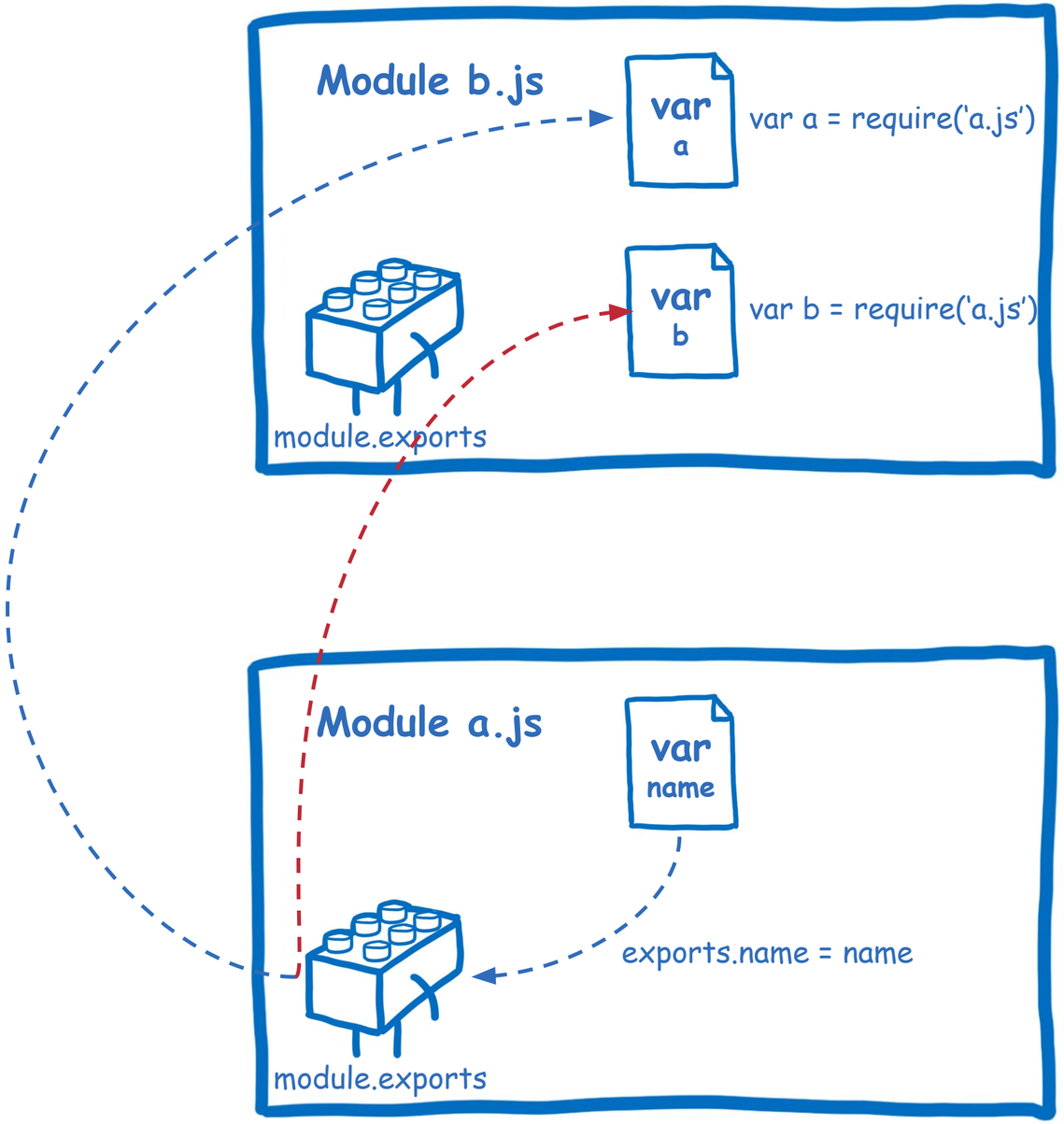


module.exports



Node.js caches the module

```
// a.js
var name = 'morrain'
var age = 18
exports.name = name
exports.getAge = function(){
    return age
}
// b.js

var a = require('a.js')
console.log(a.name) // 'morrain'
a.name = 'rename'
var b = require('a.js')
console.log(b.name) // 'rename'
                    or 'morrain'
```



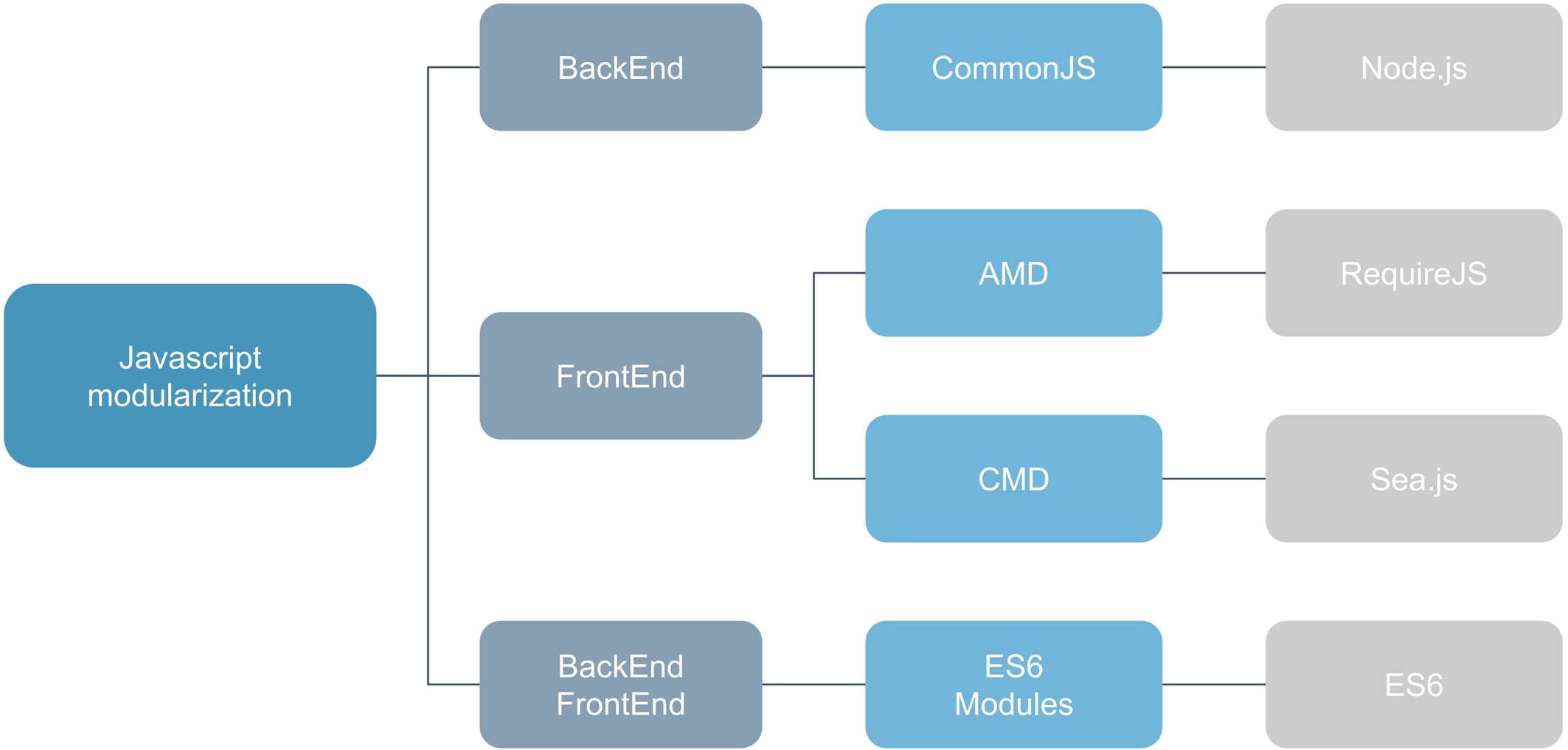
The Implementation of CommonJS

- **Each file is a module**
- **Each file has its own scope**
- **Each module has `require` and `module`**
- **`module.exports` saves the value to export**

```
(function(module, exports, require) {
    // b.js
    var a = require("a.js")
    console.log('a.name=', a.name)
    console.log('a.age=', a.getAge())

    var name = 'lilei'
    var age = 15
    exports.name = name
    exports.getAge = function () {
      return age
    }

})(module, module.exports, require)
```

```
// bundle.js
(function (modules) {
    // 模块管理的实现
})({
  'a.js': function (module, exports, require) {
    // a.js 文件内容
  },
  'b.js': function (module, exports, require) {
    // b.js 文件内容
  },
  'index.js': function (module, exports, require) {
    // index.js 文件内容
  }
})
```

```
// 模块管理的实现
var installedModules = {}
var require = function (moduleName) {
  // 如果已经加载过，就直接返回
  if (installedModules[moduleName])
     return installedModules[moduleName].exports

  // 如果没有加载，就生成一个 module，并放到 installedModules
  var module = installedModules[moduleName] = {
    moduleName: moduleName,
    exports: {}
  }
  // 执行要加载的模块
  modules[moduleName]
     .call(modules.exports, module, module.exports, require)

  return module.exports
}

return require('index.js')
```


Javascript modularization
BackEnd
CommonJS
Node.js
FrontEnd
AMD
RequireJS
CMD
Sea.js
BackEnd FrontEnd
ES6 Modules
ES6

What`s Require.js ?

RequireJS is a JavaScript file and module loader.

It is optimized for in-browser use, but it can be used in other JavaScript environments,

like Rhino and Node.

Using a modular script loader like RequireJS will improve the speed and quality of your code.

Asynchronous Module Definition

AMD

`define(id?, dependencies?, factory)`

```
// index.html
<script src="require.js"></script>
<script src="a.js"></script>
// a.js
define(function(){
    var name = 'morrain'
    var age = 18
    return {
        name,
        getAge: () => age
    }
})
// b.js
define(['a.js'], function(a){
    var name = 'lilei'
    var age = 15
    console.log(a.name) // 'morrain'
    console.log(a.getAge()) // 18
    return {
        name,
        getAge: () => age
    }
})
```

Sea.js

What`s Sea.js ？

Sea.js 追求简单、自然的代码书写和组织方式，具有以下核心特性：

简单友好的模块定义规范：Sea.js 遵循 CMD 规范，可以像 Node.js 一般书写模块代码。

自然直观的代码组织方式：依赖的自动加载、配置的简洁清晰，可以让我们更多地享受编码的乐趣。

Common Module Definition

CMD

```
define(function(require, exports, module) {

})
```

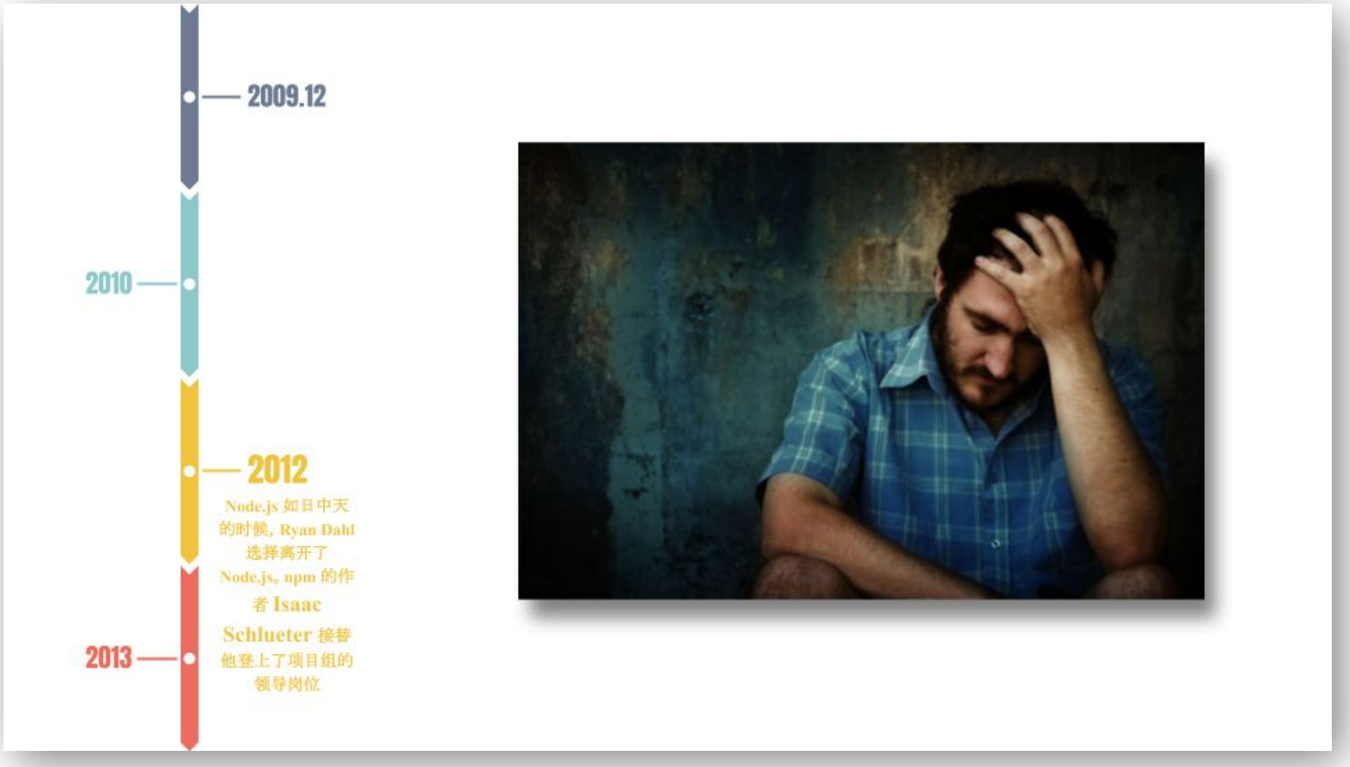
```
// index.html
<script src="sea.js"></script>
<script src="a.js"></script>
<script src="b.js"></script>
// a.js
define(function(require, exports, module){
    var name = 'morrain'
    var age = 18

    exports.name = name
    exports.getAge = () => age
})

// b.js
define(function(require, exports, module){
    var name = 'lilei'
    var age = 15
    var a = require('a.js')
    console.log(a.name) // 'morrain'
    console.log(a.getAge()) //18
    exports.name = name
    exports.getAge = () => age
})
```

ES6 Modules

Modular programming is one of the most important and frequently used software design techniques.

Unfortunately, JavaScript didn’t support modules natively that lead JavaScript programmers to use alternative techniques to achieve modular programming in JavaScript.

But now, ES6 brings modules into JavaScript officially.



"Forget CommonJS. It's dead. We are server side JavaScript."

2017-09-12, Version 8.5.0 (Current), @MylesBorins

Notable Changes

- **build**
  - Snapshots are now re-enab
- **console**
  - Implement minimal `consol`
- **deps**
  - upgrade libuv to 1.14.1 #148
  - update nghttp2 to v1.25.0 #
- **dns**
  - Add `verbatim` option to dn reshuffling that Node.js oth
- **fs**
  - add fs.copyFile and fs.copyl
- **inspector**
  - Enable async stack traces #
- **module**
  - Add support for ESM. This i `node --experimental-mo`
- **napi**
  - implement promise #14365
- **os**
  - Add support for CIDR notati
- **perf_hooks**
  - An initial implementation of the Performance Timing API for Node.js. This is the same Performance Timing API implemented by modern browsers with a number of Node.js specific properties. The User Timing mark() and

2019-11-21, Version 13.2.0 (Current), @MylesBorins

Notable Changes

- **addons**:
  - Deprecate one- and two-argument `AtExit()`. Use the three-argument variant of `AtExit()` or `AddEnvironmentCleanupHook()` instead (Anna Henningsen) #30227
- **child_process,cluster**:
  - The `serialization` option is added that allows child process IPC to use the V8 serialization API (to e.g., pass through data types like sets or maps) (Anna Henningsen) #30162
- **deps**:
  - Update V8 to 7.9
  - Update `npm` to 6.13.1 (Ruy Adorno) #30271
- **embedder**:
  - Exposes the ability to pass cli flags / options through an API as embedder (Shelley Vohr) #30466
  - Allow adding linked bindings to Environment (Anna Henningsen) #30274
- **esm**:
  - Unflag `--experimental-modules` (Guy Bedford) #29866
- **stream**:
  - Add `writable.writableCorked` property (Robert Nagy) #29012
- **worker**:
  - Allow specifying resource limits (Anna Henningsen) #26628

Using ES6 Modules in Node.js

```
// a.mjs

export const name = 'morrain'
const age = 18
export function getAge () {
    return age
}

//等价于
const name = 'morrain'
const age = 18
function getAge (){
    return age
}
export {
    name,
    getAge
}
```

```
// b.mjs

import { name as aName, getAge } from 'a.mjs'
export const name = 'lilei'
console.log(aName) // 'morrain'
const age = getAge()
console.log(age) // 18

// 等价于
import * as a from 'a.mjs'
export const name = 'lilei'
console.log(a.name) // 'morrin'
const age = a.getAge()
console.log(age) // 18
```

Using ES6 Modules in Browsers

```
<script type="module" src="./index.js">
</script>
```

```
// index.js

import { name as aName, getAge } from 'a.js'
export const name = 'lilei'
console.log(aName) // 'morrain'
const age = getAge()
console.log(age) // 18

// 等价于
import * as a from 'a.js'
export const name = 'lilei'
console.log(a.name) // 'morrin'
const age = a.getAge()
console.log(age) // 18
```

Q & A

敬请期待~~~~~

前端科普系列 之《Babel：把 ES6 送上天的通天塔》