

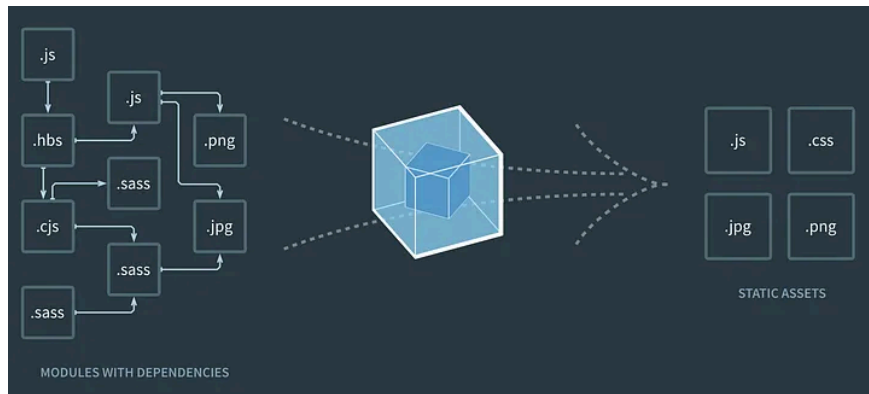
Node.js 模块系统

模块系统是现代软件系统中不可缺少的一部分。

什么是模块系统？

模块系统是指代码组织结构的一种模式，通过模块化的方式将代码分为独立的模块，以提高代码复用性和可维护性。模块系统使得大型应用程序可以被分解成小的、独立的部分，每个部分解决一个特定的问题这样做既有助于协同开发大型应用，也能够使得交付的应用程序体积更加小巧。

模块化允许代码分离，将其组织为可维护的单元，提升代码的可复用性和可读性。



webpack支持对模块打包。

为什么需要模块化?

没有模块化规范时代

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Document</title>
</head>
<body>
  <script>
    var a = 1
    var b = 2
    var c = a + b
    console.log(c)
  </script>
</body>
</html>
```

支持外联脚本的时代

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Document</title>
</head>
<body>
  <script src="./hello1.js"></script>
  <script src="./hello2.js"></script>
</body>
</html>
```

随着项目越来越大，我们需要引入的外联脚本越来越多，管理越来越困难，相互依赖关系越来越复杂，需要严格的遵循引入顺序。

没有模块化标准的问题就暴露出来了，于是 CommonJS、AMD、CMD、UMD、ESM 等模块化标准相继诞生。

Node 16 中的增补与变化

ECMAScript

5
 6
 2016+
 next
 intl
 non-standard
 compatibility table

by kangax & webbedspace & zloirock

Sort by

Engine types

 Show obsolete platforms
 ☐
 Show unstable platforms
 ☒

V8
 SpiderMonkey
 JavaScriptCore
 Chakra
 Other
 Minor difference (1 point)
 Small feature (2 points)
 Medium feature (4 points)
 Large feature (8 points)

		97%	Compilers/polyfills					Desktop browsers																Servers/runtimes							
			71%	46%	51%	5%	1%	93%	97%	98%	98%	98%	98%	98%	97%	97%	97%	94%	94%	89%	91%	94%	97%	5%	50%	81%	89%				
Feature name	Current browser		Babel 7 + core-js 3	Closure 2023.02	Type-Script + core-js 3	es7-shim	IE 11	FF 115 ESR	FF 128 ESR	FF 129	FF 130	CH 126	CH 127	Edge 126	Edge 127	SE 17.5	SF TP	WK	OP 98	OP 99	Node >=16.11 <17	Node >=18.3 <19	Node >=20 <21	Node >=22	DUK 2.7	JrS 2.4.0	GraalVM 21.3.3 ^[4]	GraalVM 22.2.0 ^[4]			
2016 features																															
• exponentiation (**) operator		3/3	3/3	3/3	2/3	0/3	0/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	2/3	3/3	3/3	3/3				
• Array.prototype.includes		4/4	4/4	3/4	4/4	3/4	0/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	0/4	4/4	4/4	4/4				
2016 misc																															
• generator functions can't be used with "new" ^[7]		Yes	?	?	?	?	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes				
• generator throw() caught by inner generator ^[8]		Yes	Yes	Yes	Yes ^[9]	?	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes				
• strict fn w/ non-strict non-simple params is error ^[10]		Yes	Yes	?	?	?	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes				
• nested rest destructuring declarations ^[11]		Yes	Yes	Yes	Yes	?	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes				
• nested rest destructuring parameters ^[12]		Yes	Yes	Yes	Yes	?	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes				
• Proxy "enumerate" handler removed ^[13]		Yes	?	?	?	?	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes				
• Proxy internal calls, Array.prototype.includes		Yes	?	?	?	?	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes				
2017 features																															
• Object static methods		4/4	4/4	3/4	4/4	3/4	0/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	0/4	4/4	4/4	4/4				
• String padding		2/2	2/2	2/2	2/2	2/2	0/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	0/2	2/2	2/2	2/2				
• trailing commas in function syntax		2/2	2/2	2/2	2/2	0/2	0/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	0/2	2/2	2/2	2/2				
• async functions		16/16	12/16	12/16	9/16	0/16	0/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	0/16	13/16	16/16	16/16				
• shared memory and atomics		12/17	0/17	0/17	0/17	0/17	0/17	17/17	17/17	17/17	17/17	17/17	17/17	17/17	17/17	17/17	17/17	17/17	17/17	17/17	17/17	17/17	17/17	0/17	0/17	17/17	17/17				
2017 misc																															
• RegExp "u" flag, case folding		Yes	?	?	?	?	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes				
• arguments.caller removed		Yes	?	?	?	?	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes				
2017 annex b																															
• Object.prototype getter/setter methods		16/16	16/16	0/16	16/16	0/16	8/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16	16/16				

CommonJS

CommonJS 是一种 JavaScript 环境中模块化编程的规范。它定义了一套模块化导入和导出的语法和机制，旨在解决 JavaScript 在模块化方面的缺陷。

导出模块

1. module.exports

```
const hello = (name) => {  
  console.log(`hello ${name}`)  
}  
  
const userInfo = {  
  name: 'forever',  
  age: 18  
}  
  
module.exports = {  
  userInfo,  
  hello  
}
```

2. exports

```
exports.hello = (name) => {  
  console.log(`hello ${name}`)  
}  
  
exports.userInfo = {  
  name: 'forever',  
  age: 18  
}
```

导入模块

1. 完整引入

```
const context = require('./exports')  
context.hello(context.userInfo.name)
```

2. 结构引入(支持)

```
const { hello: logHello, userInfo } = require('./exports')  
logHello(userInfo.name)
```

ES Module

ECMAScript 6 (ES2015/ES6) 中引入的一项重要特性，旨在取代 CommonJS 和 AMD 规范，成为 JavaScript 模块化的主要标准。

与 CommonJS 规范的区别

ESM 模块的导入和导出遵循 ECMAScript 官方规范，与 CommonJS 不同。ESM 模块的导入使用 `import` 关键字，导出使用 `export` 关键字。

默认情况下 Node.js 会将 `.js` 后缀文件识别为 CJS 模块。

要让 Node.js 正确识别，主要有两种方式：

1. 使用 `.mjs` 作为文件后缀名 (例如 `hello.mjs`)；
2. `package.json` 中 `type` 字段设置为 `module`。

导入导出

1. 普通导入导出

```
// et.mjs
export function hello(name) {
  console.log(`hello ${name}`)
}

export default {
  userInfo: {
    name: 'forever',
    age: 18
  }
}

// it.mjs
import md, {hello} from './et.mjs'

hello(
  md.userInfo.name
)
```

2. 导入导出所有对象

```
// 导出所有模块成员
export * from './et.js'

// 导入所有模块成员
import * as md from './it.mjs'
```

3. 重新导出

```
export { hello,
  default as libData
} from './lib.js'

export * from './util.js'
```

CJS与ESM的区别

	模块加载时机	导出内容的区别	文件命名
CJS	CJS 支持动态加载模块 (require 语句可以出现在任意位置)	导入的是值的拷贝	一般都以 .js 结尾
ESM	在所有模块都加载完毕后才执行代码 (通常会将 import 导入语句放在模块的顶部)	导入的是值的引用	一般都以 .mjs 结尾; 或package.json 中 "type"值为"module"