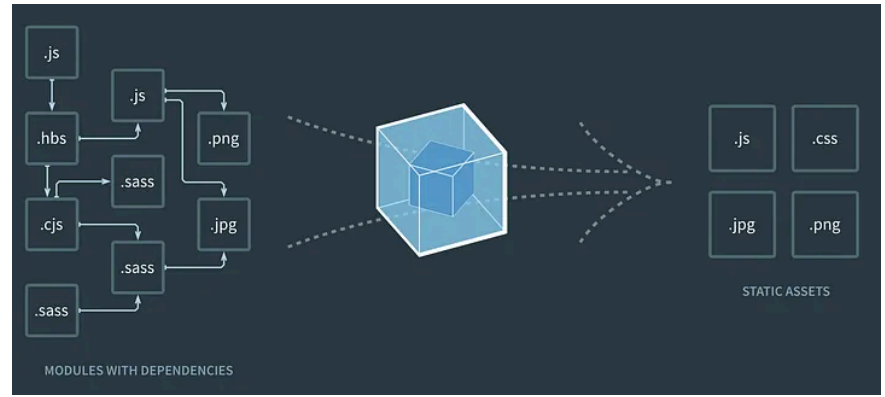
Node.js 模块系统

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.JS中的模块系统

Powered By hjp

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"