

**1.common.js**

根据CommonJS规范，一个单独的文件就是一个模块。每一个模块都是一个单独的作用域，也就是说，在该模块内部定义的变量，无法被其他模块读取，除非定义为global对象的属性

CommonJS 定义的模块分为: 模块标识(module)、模块定义(exports) 、模块引用(require)

CommonJS规范规定，每个模块内部，module变量代表当前模块。这个变量是一个对象，它的exports属性（即module.exports）是对外的接口。加载某个模块，其实是加载该模块的module.exports属性

**CommonJS 模块特点：**

1、所有代码都运行在模块作用域，不会污染全局作用域

2、模块可以多次加载，但是只会在第一次加载时运行一次，然后运行结果就被缓存了，以后再加载，就直接读取缓存结果。要想让模块再次运行，必须清除缓存

3、模块加载的顺序，按照其在代码中出现的顺序

**推荐使用import export的原因：**

1.module.exports属性表示当前模块对外输出的接口，其他文件加载该模块，实际上就是读取module.exports。 而为了方便，Node为每个模块提供一个exports变量，指向module.exports，正因为如此不能直接对exports变量赋值，因为这样会等于切断了exports与module.exports的联系。

2exports的属性和方法都可以被module.exports替代。

3.module.exports方法还可以单独返回一个数据类型，而exports返回的只能是一个对象。

所以，当我们需要返回一个数组、字符串、数字等类型时，就必须使用module.exports了。

4当使用了module.exports关键词以后，该模块中的所有exports对象执行的属性和方法都将被忽略

module.exports = ' test for module.exports ignore! '; // 返回一个字符串 exports.name = 'Galen'; /\* 定义getName函数，并暴露给外部接口 \*/

exports.getName = function() {

console.log('My name is Galen'); }; console.log( module.exports ); // 打印module.exports对象

5.所有的exports对象最终都是通过module.exports传递执行的。因此，可以说exports是给module.exports添加属性和方法

exports.name = 'a'; // 暴露name属性 exports.happy = function () { // 暴露happy方法 console.log('mm'); }; console.log( module.exports ); // 打{ name: 'a', happy: [Function] }

**加载规则**

require 命令用于加载文件，后缀名默认为.js，但如果没能查找，Node会尝试为文件名添加.js、.json、.node后，

**模块的缓存**

第一次加载某个模块时，Node会缓存该模块。所有缓存的模块保存在require.cache之中，如果想删除模块的缓存，可以这样写：

// 删除指定模块的缓存

delete require.cache[moduleName];

// 删除所有模块的缓存

Object.keys(require.cache).forEach(function(key) {

delete require.cache[key];

AMD:

使用AMD规范进行页面开发需要用到对应的库函数，也就是大名鼎鼎RequireJS

CMD:

SeaJS要解决的问题和requireJS一样，只不过在模块定义方式和模块加载（可以说运行、解析）时机上有所不同

**AMD CMD区别**

CMD和AMD解决的问题一样，不过是模块定义方式、模块加载的时机不同

/\*\* AMD写法 \*\*/ define(["a", "b", "c", "d", "e", "f"], function(a, b, c, d, e, f) { // 等于在最前面声明并初始化了要用到的所有模块 a.doSomething(); if (false) { // 即便没用到某个模块 b，但 b 还是提前执行了 b.doSomething() } }); /\*\* CMD写法 \*\*/ define(function(require, exports, module) { var a = require('./a'); //在需要时申明 a.doSomething(); if (false) { var b = require('./b'); b.doSomething(); } });

同样都是异步加载模块，AMD在加载模块完成后就会执行改模块，所有模块都加载执行完后会进入require的回调函数，执行主逻辑，这样的效果就是依赖模块的执行顺序和书写顺序不一定一致，看网络速度，哪个先下载下来，哪个先执行，但是主逻辑一定在所有依赖加载完成后才执行

CMD加载完某个依赖模块后并不执行，只是下载而已，在所有依赖模块加载完成后进入主逻辑，遇到require语句的时候才执行对应的模块，这样模块的执行顺序和书写顺序是完全一致的

这也是很多人说AMD用户体验好，因为没有延迟，依赖模块提前执行了，CMD性能好，因为只有用户需要的时候才执行的原因

**ES6: export 和 export default区别：**

export 与 export default 均可用于导出常量、函数、文件、模块等在一个文件或模块中，export、import可以有多个，export default 仅有一个通过 export 方式导出，在导入时要加 { } ， export default 则不需要export 能直接导出变量表达式， export default 不行。