# 模块化

## 什么是模块

软件工程中谈到的模块是指整个程序中一些相对对独立的程序单元，每个程序单元完成和实现一个相对独立的软件功能。通俗点就是一些功能独立的程序段。

## 如何实现模块化

由于先天的不足，ES5 以前 JS 是不支持模块化开发的。所以模块化往往多需要一些特别的写法(IIFE)完成。

## IIFE 创建简单模块

JavaScript

```
const spliter = '#'
const format = str => spliter + str + spliter
const util = {
  format
}
```

这样写模块会有什么问题。

- spliter 与 format 污染全局变量
- spliter 非私有有被篡改风险

使用立即执行函数 `IIFE（Immediately-Invoked Function Expression）` 的写法。IIFE 会创建一个只使用一次的函数，然后立即执行；IIFE 可以创建闭包进行作用域隔离，从而保护私有变量。

```JavaScript
const util = (function() {
  const spliter = '#'
  const format = str => spliter + str + spliter
  return {
    format
  }
})
```

## 前端模块化规范

模块化的实现是灵活且多样的。`我们怎么保证姿势一致和最大程度的进行模块复用呢？`这个时候就需要统一的模块化规范。

- CommonJS – nodejs
- AMD – requirejs
- CMD – sea.js
- ESM

CommonJS 规范

CommonJS 是随着 JS 在服务端的发展而发展起来的，Node.js 中的模块系统就是参照 CommonJS 规范实现的。

- 一个文件就是一个模块；
- module 对象代表模块自身，module 中有两个属性，require 和 export；
- 使用 `require(path)` 方法引入外模模块，其中 path 可以是相对路径也可以是绝对路径；
- 使用 `export` 对象作为唯一出口导出模块。

```JavaScript
// a.js
export.a = 'a';
```

```
// index.js
const moduleA = require('./a.js');
console.log(moduleA);  // {a: 'a'}
```

AMD

CommonJS 的思想是同步加载，如果在服务器端使用模块放在硬盘中性能不会有太大影响。但是在浏览器中，模块的加载需要异步加载保证性能。

这也就是 AMD （Asynchronous Module Definition） `requireJS` 是 amd 的一种实现。

CMD

`CMD(Common Module Definition)` 规范是在 SeaJs 推广过程中对模块定义的规范而产生的，也是一种在浏览器环境使用的异步模块化规范。CMD 更贴近于 `CommonJS Modules/1.1` 和 `Node Modules` 规范：

- 一个文件就是一个模块；
- 使用 define 定义模块；
- require 方法用获取其他模块提供的接口。

ESM

ESM 是 JavaScript 官方突出的标准化模块系统。在 ES 2015（ES6）中，直接在语言标准层面上实现了模块的功能。并且是浏览器和服务端都支持的模块化解决方案。

## 几种模块化规范对比

| | 运行环境 | 加载方式 | 运行机制 | 特点 | 经典实现 |
| --- | --- | --- | --- | --- | --- |
| CommonJS | 服务器 | 同步 | 运行时 | 第一次加载后会将结果缓存，再次加载会读取缓存的结构。 | NodeJS |
| AMD | 浏览器 | 异步 | 运行时 | 依赖前置，不管模块是否有用到，都会全量加载。 | RequireJS |
| CMD | 浏览器 | 异步 | 运行时 | 依赖就近，延迟加载 | SeaJS |
| ESM | 浏览器/服务端 | 异步 | 编译时 / 运行时 | 静态化，在编译时就确定模块之间的依赖关系，输入和输出。 | |