link: null
title: 珠峰架构师成长计划
description: modules\addModule.js
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=70 sentences=79, words=796

1.模块化

- 模块划就是按照一定的规则把代码封装成若干的相互依赖的文件并进行组合
- 每个模块内的数据都是私有的，只向外选择性的暴露一些方法和数据与外界进行数据通信
- 有利于代码分享、解耦以及复用
- 团队并行开发
- 避免命名冲突
- 相互引用，按需加载
- 自执行函数
- AMD (Asynchronous Module Definition)
- CMD (Common Module Definition)
- CommonJS (服务器端开发)
- UMD (Universal Module Definition)
- ES6 Module (ESM，JS官方标准模块定义方式)

2. 自执行函数

- 可能会命名冲突

```
function a(){

}
function b(){

}
```

- 减少命名冲突
- 暴露内部成员属性

```
let moduleA = {
  a(){},
  b(){}
}
let moduleB = {
  a(){},
  b(){}
}
```

- 可以保护私有变量
- 无法实现自动依赖

```
let moduleA = (function () {
    let state;
    function getState() {
        return state;
    }
    return { getState };
})();
let moduleB = (function () {
    let state;
    function getState() {
        return state;
    }
    return { getState };
})();

console.log(moduleA.getState());
console.log(moduleB.getState());
```

- 手工引入依赖
- 依赖关系不明显

```
(function (global) {
    function add(a, b) {
        return a + b;
    }
    global.addModule = { add };
})(window);
(function (global) {
    function minus(a, b) {
        return a - b;
    }
    global.minusModule = { minus };
})(window);

(function (global, addModule, minusModule) {
    global.mathModule = { add: addModule.add, minus: minusModule.minus };
})(window, addModule, minusModule);

console.log(mathModule.add(2, 2));
console.log(mathModule.minus(2, 2));
```

3. AMD

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

```
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Documenttitle>
head>
<body>
    <script type="text/javascript" src="http://static.js.xywy.com/common/js/require.min.js">script>
    [illegible]
[illegible]

[illegible]
```

```
require.config({
    baseUrl: 'modules'
});
require(['addModule', 'minusModule'], (addModule, minusModule) => {
    console.log(addModule.add(1, 2), minusModule.minus(3, 4));
});
```

modules\addModule.js

```
define(function () {
    function add(a, b) {
        return a + b;
    }
    return {
        add
    }
});
```

2.js\amd\modules\minusModule.js

```
define(function () {
    function minus(a, b) {
        return a - b;
    }
    return {
        minus
    }
});
```

amd\amd.js

```
let moduleFactory = {};
function define(name, factory) {
    moduleFactory[name] = factory;
}
function require(dependencies, callback) {
    callback(...dependencies.map(dependency => moduleFactory[dependency]()));
}

define('addModule', function () {
    function add(a, b) {
        return a + b;
    }
    return {
        add
    }
});
define('minusModule', function () {
    function minus(a, b) {
        return a - b;
    }
    return {
        minus
    }
});
require(['addModule', 'minusModule'], function (addModule, minusModule) {
    console.log(addModule.add(1, 2), minusModule.minus(3, 4));
})
```

4. CMD

- CMD 叫做通用模块定义规范（Common Module Defination ）
- CMD 规范尽量保持简单，并与 CommonJS 的 Modules 规范保持了很大的兼容性
- 通过 CMD 规范书写的模块，可以很容易在 Node.js 中运行。在 CMD 规范中，一个模块就是一个文件
- seajs (http://seajs.org/)
- CMD (https://github.com/seajs/seajs/issues/242)
- module (https://github.com/cmdjs/specification/blob/master/draft/module.md)
- **4.1 index.html**

```
<body>
  <script src="https://cdn.bootcss.com/seajs/3.0.3/sea.js">script>
  <script src="index.js">script>
  <script>
      [illegible]
  [illegible]
[illegible]
```

cmd\index.js

```
define(function (require, exports) {
    var addModule = require('./modules/addModule')
    let result1 = addModule.add(1, 2);
    console.log(result1);
    var minusModule = require('./modules/minusModule')
    let result2 = minusModule.minus(1, 2);
    console.log(result2);
})
```

modules\addModule.js

```
define(function (require, exports) {
    exports.add = function (a, b) {
        return a + b;
    }
})
```

modules\minusModule.js

```
define(function (require, exports) {
    exports.minus = function (a, b) {
        return a - b;
    }
})
```

cmd\sea.js

```
let factories = {}
let modules = {};
function require(name) {
    if (modules[name]) {
        return modules[name];
    }
    let factory = factories[name];
    let exports = {};
    factory(require, exports);
    modules[name] = exports;
    return exports;
}
function define(name, factory) {
    factories[name] = factory;
}
function use(name) {
    require(name);
}
define('addModule', function (require, exports) {
    exports.add = function (a, b) {
        return a + b;
    }
})
define('minusModule', function (require, exports) {
    exports.minus = function (a, b) {
        return a - b;
    }
})
define('index', function (require, exports) {
    var addModule = require('addModule')
    let result1 = addModule.add(1, 2);
    console.log(result1);
    var minusModule = require('minusModule')
    let result2 = minusModule.minus(1, 2);
    console.log(result2);
})
use('index');
```

5. COMMON.js

- CommonJs 是一种 JavaScript 语言的模块化规范，它通常会在服务端的 Nodejs 上使用
- 每一个文件就是一个模块，拥有自己独立的作用域、变量、以及方法等，对其他的模块都不可见
- 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（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。require 方法用于加载模块

a.js

```
module.exports = {
    a: 1
};
```

b.js

```
var a = require('./a');
var b = a.a + 2;
module.exports ={
    b
};
```

6.UMD

- UMD 叫做通用模块定义规范(Universal Module Definition)可以通过运行时或者编译时让同一个代码模块在使用 CommonJs、CMD 甚至是 AMD 的项目中运行

```
((root, factory) => {
    if (typeof define === 'function' && define.amd) {

        define(['jquery'], factory);
    } else if (typeof exports === 'object') {

        var $ = require('jquery');
        module.exports = factory($);
    } else {
        root.testModule = factory(root.jQuery);
    }
})(this, ($) => {

});
```

7.ESM

- ES6 模块的设计思想是尽量的静态化，使得编译时就能确定模块的依赖关系，以及输入和输出的变量
- ES6 Module默认目前还没有被浏览器支持，需要使用babel
- CommonJS 模块输出的是一个值的拷贝,ES6 模块输出的是值的引用
- CommonJS 模块是运行时加载，ES6 模块是编译时输出接口

```
var a = 0;
export { a };

export const b = 1;

let c = 2;
export default { c }

let d = 2;
export default { d as e }

import { a } from './a.js'

import main from './c'

import 'lodash'
```

- export {
- export default这种方式称为 默认导出 或者 匿名导出，导出的是一个值。

```
let x = 10
let y = 20
setTimeout(()=>{
    x = 100
    y = 200
},100)
export { x }
export default y

import { x } from './a.js'
import y from './a.js'
setTimeout(()=>{
    console.log(x,y)
},100)
```