Webpack原理

从前端模块化开始

nashaofu

JS模块化方案

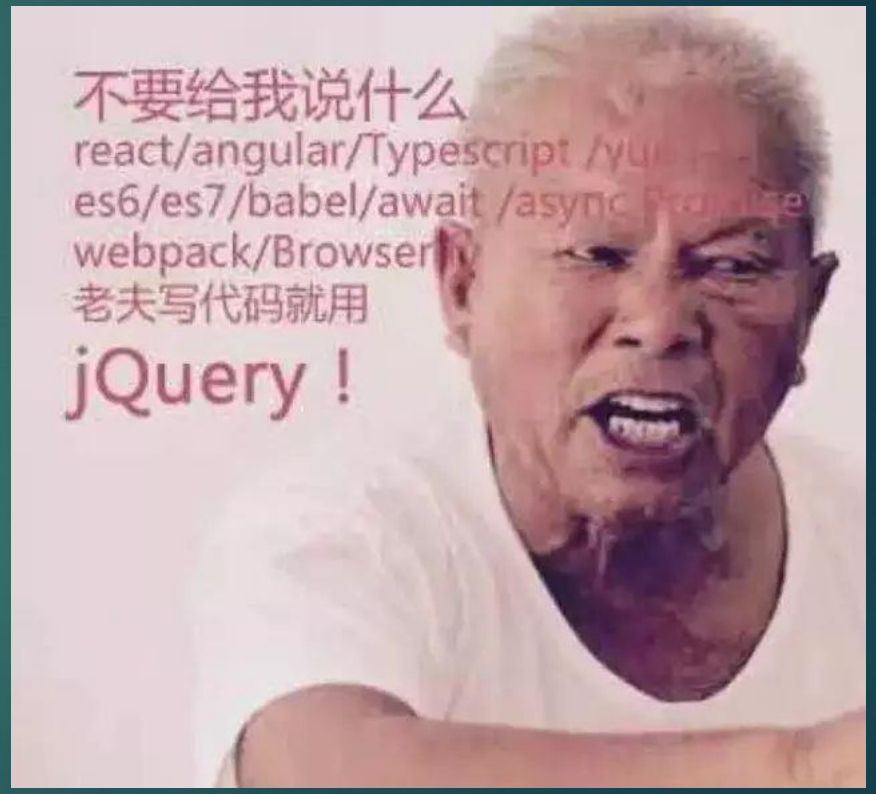
- CommonJS规范，nodejs实现的规范
- AMD规范，requirejs实现的规范
- CMD规范，seajs实现的规范， seajs与requirejs实现原理有很多相似的地方
- ES Modules，当前js标准模块化方案

注意：cjs、amd、cmd、 ES Modules都是只规范，所以可能对应有多种实现

无模块化

```
<script src="jquery.js"></script>
<script src="jquery_scroller.js"></script>
<script src="main.js"></script>
<script src="other1.js"></script>
<script src="other2.js"></script>
<script src="other3.js"></script>
```

- 污染全局作用域
- 不便于拆分逻辑，维护成本高
- 依赖关系不明显
- 复用性差



CommonJS规范

- **CommonJS是由node实现的一套规范，关于CommonJS的提出可参考** https://zhaoda.net/webpack-handbook/commonjs.html
- require源码解读可参考 http://www.ruanyifeng.com/blog/2015/05/require.html
- 模块包装相当于执行如下代码， compiledWrapper是调用node封装的V8原生创建函数的方法返回的一个函数

```
function compiledWrapper(exports, require, module, __filename, __dirname) {
 // 插入文件中的代码

 // 返回导出对象
 return module.exports
}
compiledWrapper.call(exports, exports, require, module, filename, dirname)
```

CommonJS规范

- CommonJS模块输出的是一个值的拷贝，也就是说，一旦输出一个值，模块内部的变化就影响不到这个值
- 如下有两个文件，执行命令node index.js，会有什么结果？

```
// lib.js
let counter = 3
function incCounter() {
  counter++
}

module.exports = {
 counter,
 incCounter
}
```

```
// index.js
const mod = require('./lib')

// 此处输出值？
console.log(mod.counter)
mod.incCounter()

// 此处输出值？
console.log(mod.counter)
```

CommonJS规范

- require命令第一次加载该脚本，就会执行整个脚本，然后在内存生成一个对象，下次加载会直接从缓存中取数据
- 以下是一个循环引用的例子，请问执行node main.js后会输出什么？

```
// a.js
console.log('a starting')
exports.done = false
const b = require('./b.js')
console.log('in a, b.done = %j', b.done)
exports.done = true
console.log('a done')
```

```
// b.js
console.log('b starting')
exports.done = false
const a = require('./a.js')
console.log('in b, a.done = %j', a.done)
exports.done = true
console.log('b done')
```

```
// main.js
console.log('main starting')
const a = require('./a.js')
const b = require('./b.js')
console.log('in main, a.done = %j, b.done = %j',
a.done, b.done)
```

AMD规范

- AMD是Asynchronous Module Definition的简写，即异步模块定义
- AMD规范的完整定义可参考 https://github.com/amdjs/amdjs-api/wiki/AMD
- requirejs是在浏览器中运行的，所有一些基础库需要先配置，以方便其他库调用，可以理解为CommonJS中的node_modules下的包。业务模块也可定义在其中，可认为是路径别名。paths中的路径不能包含扩展名。

```
require.config({
  paths: {
    // 如果第一个加载失败就会加载第二个
    jquery: ['lib/jquery.min', 'lib/jquery'],
    lodash: 'lib/lodash.min',
    main: './mian'  // 入口文件
  }
})
```

AMD规范

- 定义模块

```
/**
* 定义模块，当依赖加载完成后执行回调
* 回调可返回值，返回值会被导出到外部使用
* @param {String} id 模块名称，可省略
* @param {Array} dependencies 依赖的模块
* @param {Function} factory 回调函数
*/
define(id?, dependencies?, factory);
```

```
define(['jquery'], function($) {
  $('body').css({ background: 'red' })
  // 导出log函数
  return (...args) => console.log('自定义log', ...args)
})
```

AMD规范

- 加载模块

```
/**
* 加载模块
* @param {Array} deps 要加载的模块
* @param {Function} callback 加载成功回调，回调参数为加载模块导出对象
* @param {Function} errback 加载失败回调
*/
requirejs(deps, callback, errback)
```

```
require(['main'], log => {
  log('我成功加载了‘)
  // do something...,也可以在这里继续require其他js文件
})
```

requirejs 使用示例

```
.
├── index.html
└── js
    ├── lib
    │   ├── jquery.js
    │   ├── lodash.js
    │   └── require.js
    ├── main.js
    └── time.js
```

目录结构

index.html

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta http-equiv="X-UA-Compatible" content="ie=edge" />
  <title>requirejs-demo</title>
</head>
<body>
  <h1 id="time"></h1>
  <script src="./js/lib/require.js" data-main="./js/main.js"></script>
</body>
</html>
```

requirejs 使用示例

main.js

```
requirejs.config({
  baseUrl: '/js/',
  paths: {
    jquery: './lib/jquery',
    lodash: './lib/lodash'
  }
})
require(['jquery', './js/time.js'], ($, time) => {
  $('#time').text('TIME: ' + time.getTime())
  setInterval(() => {
    $('#time').text('TIME: ' + time.getTime())
  }, 1000)
})
```

time.js

```
define(['jquery', 'lodash'], ($, _) => ({
  getTime() {
    const time = new Date()
    const year = time.getFullYear()
    const month = _.padStart(time.getMonth() + 1, 2, '0')
    const date = _.padStart(time.getDate(), 2, '0')
    const hour = _.padStart(time.getHours(), 2, '0')
    const minute = _.padStart(time.getMinutes(), 2, '0')
    const second = _.padStart(time.getSeconds(), 2, '0')
    return `${year}/${month}/${date} ${hour}:${minute}:${second}`
  }
}))
```

CMD规范

- CMD是Common Module Definition的简写，即通用模块定义
- CMD规范的完整定义可参考https://github.com/seajs/seajs/issues/242
- CMD的主要代表是seajs。CMD推崇**依赖就近**，AMD推崇**依赖前置**。即**AMD在定义模块的时候就必须把依赖包含进来，CMD是在使用的时候再require对应的依赖**
- **当前主流的库对CMD支持不是很友好，都需要额外的修改才能工作**

```
// CMD
// 代码写起来有同步require的感觉
define((require, exports, module) => {
  const $ = require('jquery‘)
  $('title').text('hello')
})
```

```
// AMD
// 明显的异步风格
define(['jquery'], ($) => {
  $('title').text('hello')
})
```

seajs中require书写约定

1.正确拼require

```
// 错误！
define(function(req) {
  // ...
})

// 正确！
define(function(require) {
  // ...
})
```

2.使用直接量

```
// 错误！
require(myModule)

// 错误！
require('my-' + 'module')

// 错误！
require('MY-MODULE'.toLowerCase())

// 正确！
require('my-module')
```

```
// 错误 - 重命名 "require"！
var req = require, mod = req("./mod")

// 错误 - 重定义 "require"！
require = function() {}

// 错误 - 重定义 "require" 为函数参数！
function F(require) {}

// 错误 - 在内嵌作用域内重定义了 "require"！
function F() {
  var require = function() {}
}
```

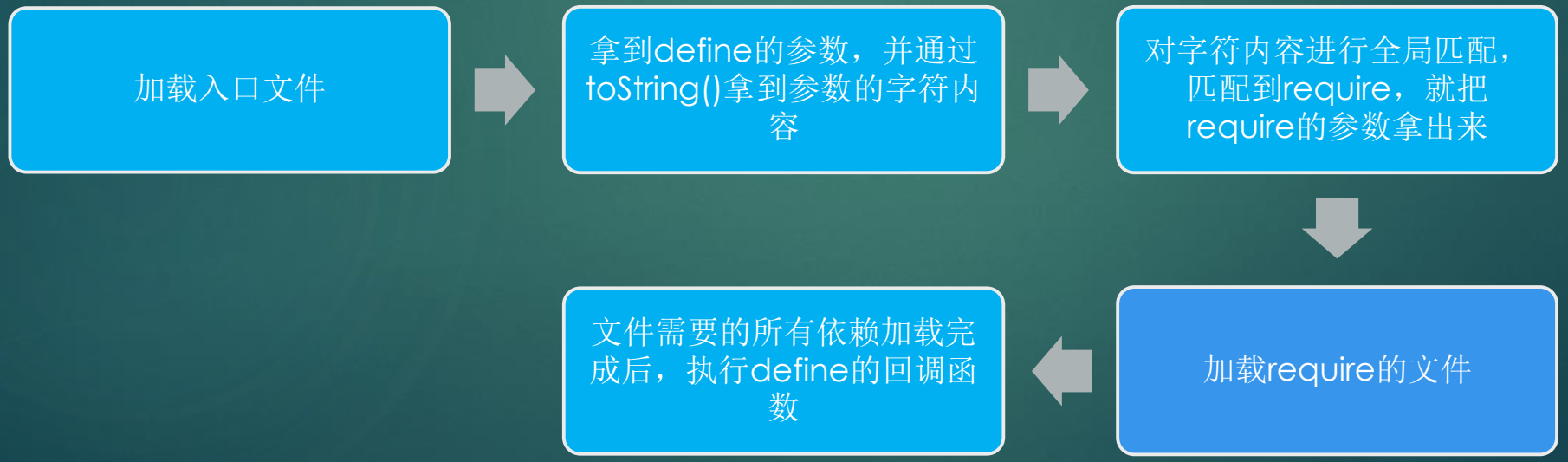
3.不要修改require

seajs隐藏坑

```
function func(require, exports, module) {
  const $ = require('jquery‘)
  console.log($)
}
func.toString = () => '() => {}'
define(func)
```

输出$为null

seajs对于require和define函数的特殊要求是由于，seajs原理导致的，seajs的执行流程大致如下



seajs使用示例

```
.
├── index.html
└── js
    ├── lib
    │   ├── jquery.js
    │   ├── lodash.js
    │   └── sea.js
    ├── main.js
    └── time.js
```

目录结构

index.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset=“UTF-8” />
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />
 <meta http-equiv="X-UA-Compatible" content="ie=edge" />
 <title>seajs-demo</title>
</head>
<body>
 <h1 id="time"></h1>
 <script src="./js/lib/sea.js" data-main="./js/main.js"></script>
 <script>
  seajs.config({
   base: '/js/',
   alias: {
    jquery: './lib/jquery',
    lodash: './lib/lodash'
   }
  })
  // 加载入口模块
  seajs.use('./js/main.js')
</script>
</body>
</html>
```

seajs使用示例

main.js

```
define((require, exports, module) => {
  const $ = require('jquery‘)
  const time = require('./time.js‘)
  $('#time').text('TIME: ' + time.getTime())
  setInterval(() => {
    $('#time').text('TIME: ' + time.getTime())
  }, 1000)
})
```

time.js

```
define((require, exports, module) => {
  module.exports = {
    getTime() {
      const $ = require('jquery‘)
      const _ = require('lodash‘)
      const time = new Date()
      const year = time.getFullYear()
      const month = _.padStart(time.getMonth() + 1, 2, '0‘)
      const date = _.padStart(time.getDate(), 2, '0‘)
      const hour = _.padStart(time.getHours(), 2, '0‘)
      const minute = _.padStart(time.getMinutes(), 2, '0‘)
      const second = _.padStart(time.getSeconds(), 2, '0‘)
      return `${year}/${month}/${date} ${hour}:${minute}:${second}`
    }
  }
})
```

ES Modules

- ES Modules是ECMAScript modules的简写，也可写为ESM。 ES Modules是js官方推出的标准
- ES Modules相比于其他模块规范是一个静态化的模块解决方案，其他模块化方案都是运行时才能确定输出内容，而ES Modules是编译时就确定了的。其他模块化方案导入文件都是整个导入模块，而ES Modules可以只导入需要的部分
- ES Modules会自动采用严格模式，不需要像ES5一样在头部加上"use strict"
- ES Modules可运行在服务端（node）和浏览器。目前主流浏览器都已经支持ES Modules，node使用ES Modules需要在执行时加上--experimental-modules，且要求编写的js文件必须以.mjs为后缀

ES Modules

- ES Modules导出的是一个值得引用，即在模块内改变了导出值，那么下一次使用也会得到新的值
- 如下有两个文件，执行命令node --experimental-modules index.mjs，会有什么结果？

```
// lib.mjs
export let counter = 3

export function incCounter () {
  counter++
}
```

```
// index.mjs
import * as mod from './lib'

// 此处输出值？
console.log(mod.counter)
mod.incCounter()

// 此处输出值？
console.log(mod.counter)
```

ES Modules

- ES Modules循环引用，请问执行node --experimental-modules main.mjs后会输出什么内容

```
// a.mjs
import { bar } from './b.mjs'
console.log('a.mjs')
console.log(bar)
export let foo = 'foo'
```

```
// b.mjs
import { foo } from './a.mjs'
console.log('b.mjs')
console.log(foo)
export let bar = 'bar'
```

```
// main.mjs
import './a.mjs'
```

requirejs与seajs

- requirejs支持seajs的CMD形式的的写法，不过requirejs作者推荐使用依赖先声明的方式。如果模块想要使用CMD的形式，那么就不能再define函数中书写依赖项，否则就会以AMD的方式进行解析。所以前面seajs例子中的模块可以直接使用requirejs加载
- 如果require加载cmd模块，也必须遵循CMD模块的编写规范，即require不能重写，define参数的toString不能重写
- 按AMD规范书写模块，性能会高于CMD形式书写的模块，因为不用执行toString方法，不用匹配代码中的require
- AMD模块对压缩友好，CMD模块必须使用特别的工具进行压缩
- CMD模块写法与CommonJS更接近
- AMD与CMD都有循环依赖问题，具体结果可自行实验

循环依赖问题

- 在所有的模块规范中都存在循环依赖问题，解决依赖循环的方式都相似，几乎都采用惰性导入的方式来解决。
- 如下两个文件存在循环引用，当执行node --experimental-modules a.mjs时，会报错说b未定义，这就是由于循环依赖导致的，如果不使用b则不会报错，修改方案如下。其他的模块循环引用也可按照此方法进行修改。
- CommonJS也可以使用先导出自身，再引入其他模块的方式尽心避免。同时也可以把require放入到函数体中，即在调用的时后才去加载依赖

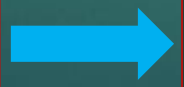
```
// a.mjs
import { b } from './b'
export const a = 1

console.log('A:', b)
```

```
// a.mjs
import { b } from './b'
export const a = 1

console.log('A:', b())
```

```
// b.mjs
import { a } from './a'
export const b = a + 1
```



```
// b.mjs
import { a } from './a'
export const b = () => a + 1
```

相关链接

- AMD 和 CMD 的区别有哪些？ - 玉伯的回答 - 知乎 https://www.zhihu.com/question/20351507/answer/14859415
- https://github.com/seajs/seajs/issues/277
- https://github.com/seajs/seajs/issues/242
- PPT中示例源码： https://github.com/nashaofu/talk-is-cheap-show-me-the-code/blob/master/webpack%E5%8E%9F%E7%90%86/%E4%BB%8E%E5%89%8D%E7%AB%AF%E6%A8%A1%E5%9D%97%E5%8C%96%E8%AE%B2%E8%B5%B7/README.md，源码示例不完善，仅供参考

关于作者

- Github： https://github.com/nashaofu
- 简书： https://www.jianshu.com/u/2bdf94072c37
- 掘金： https://juejin.im/user/594130695c497d006bb7ba21

欢迎关注nashaofu