optimization: {

removeAvailableModules: true,

removeEmptyChunks: true,

mergeDuplicateChunks: true,

//For SplitChunkss----------------------

minimize: false,

runtimeChunk: {

name: 'vendorsss'

},

splitChunks: {

cacheGroups: {

default: false,

commons: {

test: /node\_modules|layout/,

name: "vendor111",

chunks: "initial",

minSize: 1

}

}

}

},

Index.html

<script src="dist/vendor111.js"></script>

<script src="dist/vendorsss.js"></script>

<script src="dist/main.js"></script>

<script src="dist/better.js"></script>

我们来看看参数的含义：

module.exports = {

context: \_\_dirname,

entry: {

main: ["./src/main.ts"],

better: ["./src/good.js"]

},

output: {

path: path.resolve(\_\_dirname, "dist"),

publicPath: "dist",

filename: "[name].js",

//chunkFilename: 'js/[name].chunk.js'

},

optimization: {

removeAvailableModules: true,

removeEmptyChunks: true,

mergeDuplicateChunks: true,

minimize: false,

splitChunks: {

cacheGroups: {

default: false,

yep: { **--关键字只有在没有指定 name 的时候，作为文件名的一部分**

test: /node\_modules|layout/,

name: "vendor111",

chunks: "initial",

minSize: 1

},

okk: { **--关键字如果指定了name,则基本没有什么用途，只是唯一识别而已**

test: /node\_modules|home/,

name: "vendor222",

chunks: "initial",

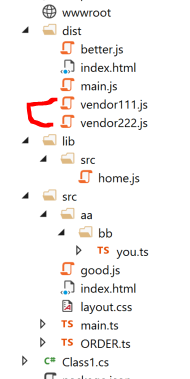
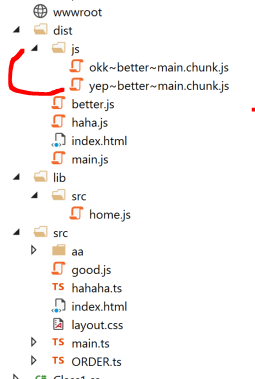
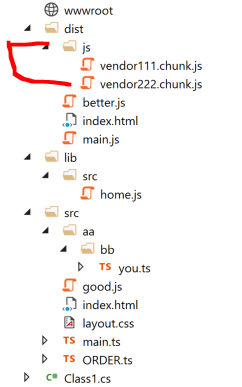
minSize: 1

}

}

}

}

output: {

path: path.resolve(\_\_dirname, "dist"),

publicPath: "dist",

filename: "[name].js",

chunkFilename: 'js/[name].chunk.js'

},

如中图： 文件名和路径由：output -> chunkFilename 来决定

splitChunks: {

cacheGroups: {

default: false,

yep: {

test: /node\_modules|layout/,

//name: "vendor111",

chunks: "initial",

minSize: 1

},

okk: {

test: /node\_modules|home/,

//name: "vendor222",

chunks: "initial",

minSize: 1

}

}

}

如右图： 关键字 ~ entry Key (relative module) …

总结：

commonChunk :

1. 如果我们定义了多个chunks 在 cacheGroups， 如果匹配模式有重复的，例如

node\_modules ，我们使用了jquery 模块。 在两个chunk 里都有。

yep: {

test: /node\_modules|layout/,

//name: "vendor111",

chunks: "initial",

minSize: 1

},

okk: {

test: /node\_modules|home/,

//name: "vendor222",

chunks: "initial",

minSize: 1

}

那么实际的情况是： 对于重复定义的， 只会在最前面的 chunk 里导入模块，其后有重复定义的chunk 里不会再导入该重复的模块。 这样就可以避免重复代码。

1. 正如commonChunk 的名字一样， 生成的所有 common chucks 是整个应用共享的chunks , 需要把所有分离出来的公共的模块一起加载到页面里。 记住既然是公共共享，缺一不可。

<script src="dist/js/vendor111.chunk.js"></script>

<script src="dist/js/vendor222.chunk.js"></script>

<script src="dist/main.js"></script>

<script src="dist/better.js"></script>

<https://webpack.toobug.net/zh-cn/chapter3/common-chunks-plugin.html>

# commonChunks插件

## 问题

顾名思义，Common Chunks 插件的作用就是提取代码中的公共模块，然后将公共模块打包到一个独立的文件中去，以便在其它的入口和模块中使用。

main.js：

var a=require('./a');

a.sayHello();

var b = require('./b');

b.sayHello();

var c = require('./c');

c.sayHello();

main.2.js：

var a=require('./a');

a.sayHello();

var b = require('./b');

b.sayHello();

a、b、c和之前一样，只有一个sayHello()方法。

打包后看到bundle.main1.js和bundle.main2.js中分别包含了a、b、c三个模块。其中的a和b正是我们要使用CommonChunksLoader提取出来的公共模块。

## 小试牛刀

接下来，我们在配置项中添加CommonChunksLoader的配置，使用它来提取公共模块。

var webpack = require('webpack');

module.exports = {

entry:{

main1:'./main',

main2:'./main.2'

},

output:{

filename:'bundle.[name].js'

},

plugins: [

new webpack.optimize.CommonsChunkPlugin('common.js', ['main1', 'main2'])

]

};

注意第1行，添加了webpack的引用（同时也要在项目目录下安装webpack），然后添加了plugins选项，引用了webpack.optimize.CommonsChunkPlugin来提取公共模块，参数common.js表示公共模块的文件名，后面的数组元素与entry一一对应，表示要提取这些模块中的公共模块。

重新使用webpack打包后，看到生成的文件中多了一个common.js：

webpack

Hash: 2eaa6808a94e7ed42693

Version: webpack 1.12.8

Time: 99ms Asset Size Chunks Chunk Names

bundle.main1.js 417 bytes 0 [emitted] main1

bundle.main2.js 193 bytes 1 [emitted] main2

common.js 3.9 kB 2 [emitted] common.js

[0] ./main.js 117 bytes {0} [built]

[0] ./main.2.js 77 bytes {1} [built]

[1] ./a.js 97 bytes {2} [built]

[2] ./b.js 97 bytes {2} [built]

[3] ./c.js 97 bytes {0} [built]

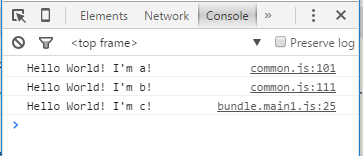
这个common.js正是公共部分a和b两个模块。而生成的bundle.main1.js中只包含了c模块，bundle.main2.js中则没有包含任何其它模块。生成后的代码可参见[https://github.com/TooBug/webpack-guide/blob/master/examples/chapter3/common-chunks-plugin](https://github.com/TooBug/webpack-guide/blob/master/examples/chapter3/common-chunks-plugin" \t "_blank)。

当然，还有一步不能少，就是在HTML中加入公共部分common.js：

<script src="./common.js"></script>

<script src="./bundle.main1.js"></script>

运行时截图如下：



## 另一个问题

上面的代码成功地将公共模块a和b提取出来了。但不知道看到此处，你的心里是否会有点疑问。

如果你还没有的话，我们一起来看一下下面这些东东：

1. 什么是入口文件？如果你还记得前面我们给出的定义的话，就会发现，这个地方有两个入口文件！
2. 模块化方案应该允许多个入口文件吗？应该吗？不应该吗？应该吗？不应该吗？
3. 如果模块规划调整了，common.js消失了，或者增加了一个common.2.js，我应该修改每一个HTML引入的部分吗？

其实写到这里，我是有点迷茫的，按照我对模块化方案的认知，我的观点是：

1. 入口文件应该只有一个，其它的逻辑全部由入口文件处理，因此不应该允许多入口文件
2. 模块规划应该在模块化方案内部完成，不应该还需要时时调整模块调用入口

<http://www.itboth.com/d/u6vYZf/webpack>

### Webpack 4.0 不再使用CommonsChunkPlugin

webpack 4.0.0-beta.0删除了 CommonsChunkPlugin ，以支持两个新的选项（ optimization.splitChunks 和 optimization.runtimeChunk ）。

从webpack 4.0.0-beta.0 开始分割 Chunk 将不在使用 CommonsChunkPlugin 插件，而是使用 optimization 配置项，具体的实现原理可以参考 [CommonsChunkPlugin](https://gist.github.com/sokra/1522d586b8e5c0f5072d7565c2bee693) 。

由于还没有正式官方文档出来，以下是我们通过实践出的 optimization 配置方法：

其中用到了新增的 splitChunks 属性，此属性看字面意思就明白是分割代码块的选项，其下可配置项已在下面示例代码中列出（有兴趣的朋友可以自行实践）：

entry: {

     vendor: ['lodash']

},

...

optimization: {

     splitChunks: {

     chunks: "initial", // 必须三选一： "initial" | "all"(默认就是all) | "async"

     minSize: 0, // 最小尺寸，默认0

     minChunks: 1, // 最小 chunk ，默认1

     maxAsyncRequests: 1, // 最大异步请求数， 默认1

     maxInitialRequests : 1, // 最大初始化请求书，默认1

     name: function(){}, // 名称，此选项可接收 function

     cacheGroups:{ // 这里开始设置缓存的 chunks

         priority: 0, // 缓存组优先级

         vendor: { // key 是唯一识别号，如果没有指定name, 那么会作为chunk 名称的一部分

             chunks: "initial", // 必须三选一： "initial" | "all" | "async"(默认就是异步)

             test: /react|lodash/, // 正则规则验证，如果符合就提取 chunk

             name: "vendor", // 要缓存的 分隔出来的 chunk 名称

             minSize: 0,

             minChunks: 1,

             enforce: true,

             maxAsyncRequests: 1, // 最大异步请求数， 默认1

             maxInitialRequests : 1, // 最大初始化请求书，默认1

             reuseExistingChunk: true // 可设置是否重用该chunk（查看源码没有发现默认值）

         }

     }

  }

},

以上就是 optimization.splitChunks 的所有可用的配置项属性。

<https://blog.csdn.net/qq_16559905/article/details/79404173>

optimization参数介绍：

optimization: {

splitChunks: {

chunks: "initial", // 必须三选一： "initial" | "all"(默认就是all) | "async"

minSize: 0, // 最小尺寸，默认0

minChunks: 1, // 最小 chunk ，默认1

maxAsyncRequests: 1, // 最大异步请求数， 默认1

maxInitialRequests: 1, // 最大初始化请求书，默认1

name: () => {}, // 名称，此选项课接收 function

cacheGroups: { // 这里开始设置缓存的 chunks

priority: "0", // 缓存组优先级 false | object |

vendor: { // key 为entry中定义的 入口名称

chunks: "initial", // 必须三选一： "initial" | "all" | "async"(默认就是异步)

test: /react|lodash/, // 正则规则验证，如果符合就提取 chunk

name: "vendor", // 要缓存的 分隔出来的 chunk 名称

minSize: 0,

minChunks: 1,

enforce: true,

maxAsyncRequests: 1, // 最大异步请求数， 默认1

maxInitialRequests: 1, // 最大初始化请求书，默认1

reuseExistingChunk: true // 可设置是否重用该chunk（查看源码没有发现默认值）

}

}

}

},

==最后，optimization 使用相关内容在这里==

[CommonsChunkPlugin](https://gist.github.com/sokra/1522d586b8e5c0f5072d7565c2bee693?utm_source=aotu_io&utm_medium=liteo2_web)

<https://news.aotu.io/a/5a7b53d3d50eee0042c20c0c?utm_medium=lite02_web&utm_source=aotu_io>