# 解析(Resolve)

resolve: {

modules:["lib/src", "node\_modules"], - 搜索模块的路径，

alias: {

haohao: path.join(\_\_dirname , '/src/aa/bb')

},

mainFiles: ["you.ts", "index.js"],

// import "home" 不指定后缀名时， 则自动添加。 如 home.js

extensions: [".d.ts", ".css", ".js", ".ts"]

},

modules: 搜索模块的路径, 如果不设置默认是 “node\_modules”

 modules: "lib/src"

import "home"; -- 可以省略路径，因为我们指定了搜索路径："lib/src"

home.js: alert("this home page");

alias : "haohao" 默认的主文件： "you.ts" 所以下面代码工作正常

import { NIHAO } from "haohao";

let nn = new NIHAO("Miller in Good");

nn.show();

you.ts :

export class NIHAO {

constructor(public name: string) {}

public show(): void {

alert("NIHAO SHOW " + this.name);

}

}

import { NIHAO } from "haohao";

import "home";

import xxx from “xxxx” vs import “xxxx” 的区别：

1. Import “xxxx”

我们看看下面的例子：

home.js

----------------------------------

var ADDR = "Burlington Ave";

const YES = "yes";

alert(ADDR + ":" + YES);

---------------------------------

main.ts

-----------------------------------

import "home";

console.log(YES); -- 出错，编译不能通过

console.log(ADDR); -- 出错， 编译不能通过

import "home";

alert(ADDR + ":" + YES); -- 此语句可以成功执行， 而且变量也可以正确输出。

如果

home.js

(function (a, b) {

console.log("a & b in home!!!");

return {

name1: a,

name2: b

};

})("Hello", "World");

Console.log – 语句会被执行；

let nm = require("home"); 但是函数的返回值，不会被返回。所以 nm 的值是 {} 空对象，并非 undefined/null

console.log("People Name");

console.log(nm); -- 输出 {}

1. Import xxx from “xxx” 相对比较规矩：

如果模块里面有直接执行的语句则 同上原理，

另外模块里使用 export 输出的东西， 都可以在 import 里载入后被使用。

Webpack 性能 另外一个重要的选项： noParse

module.noParse   
RegExp | [RegExp]

RegExp | [RegExp] | function（从 webpack 3.0.0 开始）

防止 webpack 解析那些任何与给定正则表达式相匹配的文件。忽略的文件中不应该含有 import, require, define 的调用，或任何其他导入机制。忽略大型的 library 可以提高构建性能。

noParse: /jquery|lodash/

// 从 webpack 3.0.0 开始   
noParse: function(content) {   
return /jquery|lodash/.test(content);   
}   
不解析这些library

entry: {

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

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

},

module: {

rules: [

{ test: /\.ts$/, use: ["ts-loader"], exclude: /node\_modules/ },

{ test: /\.js$/, use: ["babel-loader"], exclude: /node\_modules/ },

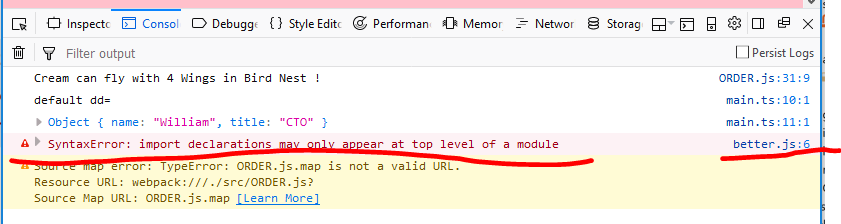
{ test: /\.css$/, use:["style-loader", "css-loader"], exclude: /node\_modules/ }

],

noParse: /jquery|home|good/

},

注意： noParse 使用正则表达式来匹配文件，如果是noParse 的文件， 里面不能有导入模块的语句如： import , require 等。否则会出错 good



如上：错误

good.js 包含有 import 语句

import "./layout.css";

import { NIHAO } from "haohao";

let nn = new NIHAO("Miller in Good");

nn.show();

$(function () {

$("body").append("<h2 class='text'>This is from Good.js</h2>");

$("body").append("<h2>This is from Good.js</h2>");

})

<http://www.css88.com/doc/webpack/configuration/resolve/>

# 解析(Resolve)

这些选项能设置模块如何被解析。webpack 提供合理的默认值，但是还是可能会修改一些解析的细节。 关于 resolver 具体如何工作的更多解释说明，请查看[模块解析方式](http://www.css88.com/doc/webpack/concepts/module-resolution)。

## resolve

object

配置模块如何解析。例如，当在 ES2015 中调用 import "lodash"，resolve 选项能够对 webpack 查找 "lodash" 的方式去做修改（查看[模块](http://www.css88.com/doc/webpack/configuration/resolve/" \l "resolve-modules)）。

### resolve.alias

object

创建 import 或 require 的别名，来确保模块引入变得更简单。例如，一些位于 src/ 文件夹下的常用模块：

alias: {

Utilities: path.resolve(\_\_dirname, 'src/utilities/'),

Templates: path.resolve(\_\_dirname, 'src/templates/')

}

现在，替换「在导入时使用相对路径」这种方式，就像这样：

import Utility from '../../utilities/utility';

你可以这样使用别名：

import Utility from 'Utilities/utility';

也可以在给定对象的键后的末尾添加 $，以表示精准匹配：

alias: {

xyz$: path.resolve(\_\_dirname, 'path/to/file.js')

}

这将产生以下结果：

import Test1 from 'xyz'; // 精确匹配，所以 path/to/file.js 被解析和导入

import Test2 from 'xyz/file.js'; // 精确匹配，触发普通解析

下面的表格展示了一些其他情况：

| **别名：** | **import "xyz"** | **import "xyz/file.js"** |
| --- | --- | --- |
| {} | /abc/node\_modules/xyz/index.js | /abc/node\_modules/xyz/file.js |
| { xyz: "/abs/path/to/file.js" } | /abs/path/to/file.js | error |
| { xyz$: "/abs/path/to/file.js" } | /abs/path/to/file.js | /abc/node\_modules/xyz/file.js |
| { xyz: "./dir/file.js" } | /abc/dir/file.js | error |
| { xyz$: "./dir/file.js" } | /abc/dir/file.js | /abc/node\_modules/xyz/file.js |
| { xyz: "/some/dir" } | /some/dir/index.js | /some/dir/file.js |
| { xyz$: "/some/dir" } | /some/dir/index.js | /abc/node\_modules/xyz/file.js |
| { xyz: "./dir" } | /abc/dir/index.js | /abc/dir/file.js |
| { xyz: "modu" } | /abc/node\_modules/modu/index.js | /abc/node\_modules/modu/file.js |
| { xyz$: "modu" } | /abc/node\_modules/modu/index.js | /abc/node\_modules/xyz/file.js |
| { xyz: "modu/some/file.js" } | /abc/node\_modules/modu/some/file.js | error |
| { xyz: "modu/dir" } | /abc/node\_modules/modu/dir/index.js | /abc/node\_modules/dir/file.js |
| { xyz: "xyz/dir" } | /abc/node\_modules/xyz/dir/index.js | /abc/node\_modules/xyz/dir/file.js |
| { xyz$: "xyz/dir" } | /abc/node\_modules/xyz/dir/index.js | /abc/node\_modules/xyz/file.js |

如果在 package.json 中定义，index.js 可能会被解析为另一个文件。

/abc/node\_modules 也可能在 /node\_modules 中解析。

### resolve.aliasFields

string

指定一个字段，例如 browser，根据[此规范](https://github.com/defunctzombie/package-browser-field-spec" \t "_blank)进行解析。默认：

aliasFields: ["browser"]

### resolve.cacheWithContext

boolean (since webpack 3.1.0)

If unsafe cache is enabled, includes request.context in the cache key. This option is taken into account by the [enhanced-resolve](https://github.com/webpack/enhanced-resolve/) module. Since webpack 3.1.0 context in resolve caching is ignored when resolve or resolveLoader plugins are provided. This addresses [a](http://www.css88.com/doc/webpack/glossary.html#a) performance regression.

### resolve.descriptionFiles

array

用于描述的 JSON 文件。默认：

descriptionFiles: ["package.json"]

### resolve.enforceExtension

boolean

如果是 true，将不允许无扩展名(extension-less)文件。默认如果 ./foo 有 .js 扩展，require('./foo') 可以正常运行。但如果启用此选项，只有 require('./foo.js') 能够正常工作。默认：

enforceExtension: false

### resolve.enforceModuleExtension

boolean

对模块是否需要使用的扩展（例如 loader）。默认：

enforceModuleExtension: false

### resolve.extensions

array

自动解析确定的扩展。默认值为：

extensions: [".js", ".json"]

能够使用户在引入模块时不带扩展：

import File from '../path/to/file'

[W](http://www.css88.com/doc/webpack/glossary.html#w)> 使用此选项，会**覆盖默认数组**，这就意味着 webpack 将不再尝试使用默认扩展来解析模块。对于使用其扩展导入的模块，例如，import SomeFile from "./somefile.ext"，要想正确的解析，一个包含“\*”的字符串必须包含在数组中。

### resolve.mainFields

array

当从 npm 包中导入模块时（例如，import \* as D3 from "d3"），此选项将决定在 package.json 中使用哪个字段导入模块。根据 webpack 配置中指定的 [target](http://www.css88.com/doc/webpack/concepts/targets) 不同，默认值也会有所不同。

当 target 属性设置为 webworker, web 或者没有指定，默认值为：

mainFields: ["browser", "module", "main"]

对于其他任意的 target（包括 node），默认值为：

mainFields: ["module", "main"]

例如，[D3](https://d3js.org/" \t "_blank) 的 package.json 含有这些字段：

{

...

main: 'build/d3.Node.js',

browser: 'build/d3.js',

module: 'index',

...

}

这意味着当我们 import \* as D3 from "d3"，实际从 browser 属性解析文件。在这里 browser 属性是最优先选择的，因为它是 mainFields 的第一项。同时，由 webpack 打包的 Node.js 应用程序默认会从 module 字段中解析文件。

### resolve.mainFiles

array

解析目录时要使用的文件名。默认：

mainFiles: ["index"]

### resolve.modules

array

告诉 webpack 解析模块时应该搜索的目录。

绝对路径和相对路径都能使用，但是要知道它们之间有一点差异。

通过查看当前目录以及祖先路径（即 ./node\_modules, ../node\_modules 等等），相对路径将类似于 Node 查找 'node\_modules' 的方式进行查找。

使用绝对路径，将只在给定目录中搜索。

resolve.modules defaults to:

modules: ["node\_modules"]

如果你想要添加一个目录到模块搜索目录，此目录优先于 node\_modules/ 搜索：

modules: [path.resolve(\_\_dirname, "src"), "node\_modules"]

### resolve.unsafeCache

regex array boolean

启用，会主动缓存模块，但并**不安全**。传递 true 将缓存一切。默认：

unsafeCache: true

正则表达式，或正则表达式数组，可以用于匹配文件路径或只缓存某些模块。例如，只缓存 utilities 模块：

unsafeCache: /src\/utilities/

[W](http://www.css88.com/doc/webpack/glossary.html#w)> 修改缓存路径可能在极少数情况下导致失败。

## resolve.plugins

应该使用的额外的解析插件列表。它允许插件，如 [DirectoryNamedWebpackPlugin](https://www.npmjs.com/package/directory-named-webpack-plugin)。

plugins: [new DirectoryNamedWebpackPlugin()]

## resolve.symlinks

boolean

是否将符号链接(symlink)解析到它们的符号链接位置(symlink location)。默认：

symlinks: true

## resolve.cachePredicate

function

决定请求是否应该被缓存的函数。函数传入一个带有 path 和 request 属性的对象。默认：

cachePredicate: function() { return true }

## resolveLoader

object

这组选项与上面的 resolve 对象的属性集合相同，但仅用于解析 webpack 的 [loader](http://www.css88.com/doc/webpack/concepts/loaders) 包。默认：

{

modules: ["node\_modules"],

extensions: [".js", ".json"],

mainFields: ["loader", "main"]

}

[T](http://www.css88.com/doc/webpack/glossary.html#t)> 注意，这里你可以使用别名，并且其他特性类似于 resolve 对象。例如，{ txt: 'raw-loader' } 会使用 raw-loader 去 shim(填充) txt!templates/demo.txt。

## resolveLoader.moduleExtensions

array

在解析模块（例如，loader）时尝试使用的扩展。默认是一个空数组。

如果你想要不带 -loader 后缀使用 loader，你可以使用：

moduleExtensions: ['-loader']