



专栏 / 不长写的日志 / 文章详情



儿独 RP 270 发布于 不长写的日志  
2017-11-22 发布

## webpack externals 深入理解

按照官方文档的解释，如果我们想引用一个库，但是又不想让webpack打包，并且又不影响我们在程序中以CMD、AMD或者window/global全局等方式进行使用，那就可以通过配置externals。这个功能主要是用在创建一个库的时候用的，但是也可以在我们项目开发中充分使用。

假设：我们开发了一个自己的库，里面引用了lodash这个包，经过webpack打包的时候，发现如果把这个lodash包打入进去，打包文件就会非常大。那么我们就可以externals的方式引入。也就是说，自己的库本身不打包这个lodash，需要用户环境提供。

使用lodash

```
import _ from 'lodash';
```

配置externals

```
externals: {  
  "lodash": {  
    commonjs: "lodash", // 如果我们的库运行在Node.js环境中，import _ from 'lodash'等价于const _ = re  
    commonjs2: "lodash", // 同上  
    amd: "lodash", // 如果我们的库使用require.js等加载，等价于 define(["lodash"], factory);  
    root: "_" // 如果我们的库在浏览器中使用，需要提供一个全局的变量'_'，等价于 var _ = (window._) or (_);  
  }  
}
```

总的来说，externals配置就是为了使 `import _ from 'lodash'` 这句代码，在本身不引入lodash的情况下，能够在各个环境都能解释执行。

有一点需要注意的是，假如lodash中在浏览器环境中不提供 `_` 的全局变量，那么就没有办法使用。这个 `"_"` 是不能随便乱写的。如果外部库lodash提供的是全局变量 `lodash`，那你就得使用 `lodash`。

如果你写的库要支持各种环境，你需要设置output中的libraryTarget为umd，也就是将打包的文件，生成为umd规范，适用于各种环境。libraryTarget和externals有藕断丝连的关系，后面会提到。

下面进入正题，externals的配置有以下几种：array, object, reg。这三种形式都可以传入，前者其实是对后者的包含。

[参考这里](#)

## Array

数组内的每一个元素又可以是多种形式, 包括object, reg, function, string

```
externals: [
  { // ① object形式
    jquery: 'jQuery', //
    a: false, // 不是external, 配置错误
    b: true, // b 是 external, `module.exports = b`, 适用于你所引用的库暴露出的变量和你所使用的库的
    "./c": "c", // "./c" 是 external `module.exports = c`
    "./d": "var d", // "./d" 是 external `module.exports = ./d` 语法错误
    "./f": "commonjs2 ./a/b", // "./f" 是 external `module.exports = require("./a/b")`
    "./f": "commonjs ./a/b", // ...和 commonjs2一样
    "./f": "this ./a/b", // "./f" 是 external `(function() { module.exports = this["./a/b"]
  },
  // abc -> require("abc")
  /^[a-z\-\0-9]+$/, // ② reg形式
  function(context, request, callback) { // ③ function形式
    // Every module prefixed with "global-" becomes external
    // "global-abc" -> abc
    if(/^global-/i.test(request))
      return callback(null, "var " + request.substr(7));
    callback();
  },
  "./e" // "./e" 是 external ( require("./e") ) // ④ string形式
]
```

## Object

Object形式和上面类似, 但是它里面一定是key: value的形式, 所以像上面那种string的形式就不可能出现在object形式中。这种情况下使用的最多。

```
externals:{
  jquery: 'jQuery', //
  a: false, // 不是external, 配置错误
  b: true, // b 是 external, `module.exports = b`, 适用于你所引用的库暴露出的变量和你所使用的库的名
  "./c": "c", // "./c" 是 external `module.exports = c`
  "./d": "var d", // "./d" 是 external `module.exports = ./d` 语法错误
  "./f": "commonjs2 ./a/b", // "./f" 是 external `module.exports = require("./a/b")`
  "./f": "commonjs ./a/b", // ...和 commonjs2一样
  "./f": "this ./a/b", // "./f" 是 external `(function() { module.exports = this["./a/b"]
},
```

reg就不介绍了, 也就是正则匹配的形式。可以类比Array类型中的string。

externals引入jquery后, 那么不管在代码中使用 `import $ from 'jquery'` 还是 `var $ = require('jquery')`; 这些代码都能在浏览器中很好的执行。这很好的验证了使用externals的情况。

想引用一个库, 但是又不想让webpack打包, 并且又不影响我们在程序中以CMD、AMD或者window/global全局等方式进行使用

那如果想要这样使用 `import $ from 'jquery'`，并且想在Node环境中使用，那么就必须要使用这样 `jquery: 'commonjs2 jquery'` 使用。这样webpack就会把你所需要的模块打包成 `module.exports = require('jquery')`，可以再Node环境中使用。

`externals` 支持以下模块上下文(module context)

- **global** - 外部 library 能够作为全局变量使用。用户可以通过在 `script` 标签中引入来实现。这是 `externals` 的默认设置。
- **commonjs** - 用户(consumer)应用程序可能使用 CommonJS 模块系统，因此外部 library 应该使用 CommonJS 模块系统，并且应该是一个 CommonJS 模块。
- **commonjs2** - 类似上面几行，但导出的是 `module.exports.default`。
- **amd** - 类似上面几行，但使用 AMD 模块系统。

## 不同环境设置externals方式

1. 如果你的代码想运行在Node环境中，那么你需要在external中添加前缀commonjs2或者commonjs

```
externals:{  
  react:'commonjs2 react',  
  jquery:'commonjs2 jquery'  
}
```

1. 如果需要requirejs等符合AMD规范的环境中加载，那就要添加amd

```
externals:{  
  react:'amd React',  
  jquery:'amd jQuery'  
}
```

1. 如果要在浏览器中运行，那么不用添加什么前缀，默认设置就是global。

```
externals:{  
  react:'React',  
  jquery:'jQuery'  
}
```

也可以这样

```
externals:["React","jQuery"]
```

这种方式配置下，就是配置你所引用你的库暴露出的全局变量。上面两种模式下或者说，如果你想运行代码在浏览器中，你所引用的包，必须暴露出一个全局变量。如果没有，这种方式不适合在浏览器下使用，可以尝试dll的方式。

这里你可以看出，不同模式下，value是不一样的。2，3模式下，是要引入去全局变量，1模式是要加载包名。那如果这个包的包名和在浏览器下引入的全局变量一致，上面就可以写成一样了，比如moment。

## externals 和 libraryTarget 的关系

- libraryTarget配置如何暴露 library。如果不设置library,那这个library就不暴露。就相当于一个自执行函数
- externals是决定的是以哪种模式去加载所引入的额外的包
- libraryTarget决定了你的library运行在哪个环境，哪个环境也就决定了你哪种模式去加载所引入的额外的包。也就是说，externals应该和libraryTarget保持一致。library运行在浏览器中的，你设置externals的模式为commonjs，那代码肯定就运行不了了。
- 如果是应用程序开发，一般是运行在浏览器环境libraryTarget可以不设置，externals默认的模式是global，也就是以全局变量的模式加载所引入外部的库。

参考：

<http://www.css88.com/doc/webp...>

<http://www.css88.com/doc/webp...>

<http://www.tangshuang.net/334...>



赞 | 13

收藏 | 18

广告

你可能感兴趣的

广告

你可能感兴趣的文章

- [vue/webpack 引入 cdn 资源](#) big\_cat cdn webpack vue-cli
- [使用webpack的插件DllPlugin加快打包速度](#) SHERlocked93 webpack 打包
- [webpack 初探](#) 蓝胖子又叫叮当猫 webpack-dev-server webpack
- [配置webpack中externals来减少打包后vendor.js的体积](#) darkerXi axios vue-router element-ui vue.js webpack
- [vue-cli脚手架中webpack配置基础文件详解](#) 切图妞 前端 vue.js webpack 配置文件