**Node 环境变量 process.env.NODE\_ENV 之webpack应用**

对于process.env.NODE\_ENV困惑起因为在配置webpack时，明明配置了：process.env.NODE\_ENV为production，如下：

new webpack.DefinePlugin({

'process.env.NODE\_ENV': JSON.stringify('production')

}),

new webpack.optimize.UglifyJsPlugin({

compress: process.env.NODE\_ENV === 'production'

})

但最终并没有生效，得到的 compress:false，本着有困难就gg的原则，找找找各种文档，总算理清了其中的关系。

DefinePlugin 在原始的源码中执行查找和替换操作，在导入的代码中，任何出现 process.env.NODE\_ENV的地方都会被替换为”production”。因此，形如if (process.env.NODE\_ENV !== ‘production’) console.log(‘…’) 的代码就会等价于 if (false) console.log(‘…’) 并且最终通过UglifyJS等价替换掉。

也就是说，webpack config文件中定义的变量：

new webpack.DefinePlugin({

'process.env.NODE\_ENV': JSON.stringify('production')

})

• 1

• 2

• 3

是为了你将要打包的文件中用的。

那如何在webpack config文件中使用 process.env.NODE\_ENV 呢？答案是corss-env

进入主题

如上信息看着也许很乱，那通过一个实例来一步步验证一下吧~

第一步：

npm init

生成一下package.json，然后install需要的模块：

npm install –save-dev webpack clean-webpack-plugin html-webpack-plugin webpack-merge cross-env

一、接下来进入主题，开始配置 webpack.config.js:

const path = require('path')

const HtmlWebpackPlugin = require('html-webpack-plugin')

const config = {

entry:path.join(\_\_dirname,"src/app.js"),

output:{

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

filename:"bundle.js"

},

plugins:[

new HtmlWebpackPlugin()

]

}

module.exports = config

console.log("process.env.NODE\_ENV 的值是(webpack.config.js)："+ process.env.NODE\_ENV)

• 1

• 2

• 3

• 4

• 5

• 6

• 7

• 8

• 9

• 10

• 11

• 12

• 13

• 14

• 15

• 16

然后新建文件 src/app.js

console.log("app test")

console.log("process.env.NODE\_ENV 的值是(app.js)："+ process.env.NODE\_ENV)

• 1

• 2

• 3

一切准备好后，给package.json加一行 ：“build”:”webpack”

"scripts": {

"build":"webpack"

}

• 1

• 2

• 3

然后执行：npm run build，控制台会有如下输出：

λ npm run build

> node@1.0.0 build D:\works\test\VUE\NODE\_ENV

> webpack

process.env.NODE\_ENV 的值是(webpack.config.js)：undefined

Hash: f323df4deba684f859b8

Version: webpack 3.4.1

Time: 589ms

Asset Size Chunks Chunk Names

bundle.js 8.18 kB 0 [emitted] main

index.html 182 bytes [emitted]

[0] ./src/app.js 97 bytes {0} [built]

+ 1 hidden module

Child html-webpack-plugin for "index.html":

[2] (webpack)/buildin/global.js 509 bytes {0} [built]

[3] (webpack)/buildin/module.js 517 bytes {0} [built]

+ 2 hidden modules

• 1

• 2

• 3

• 4

• 5

• 6

• 7

• 8

• 9

• 10

• 11

• 12

• 13

• 14

• 15

• 16

• 17

• 18

由于没有进行任何配置，所以上面的输出中给出的信息是：

process.env.NODE\_ENV 的值是(webpack.config.js)：undefined

通过浏览器访问/dist/index.html，控制台有如下信息输出：

app test

process.env.NODE\_ENV 的值是(app.js)：undefined

也就是说，在/src/app.js里，process.env.NODE\_ENV 也未定义

二、通过webpack -p参数控制

在package.json里增加一行：

"scripts": {

"build":"webpack",

"build-p":"webpack -p"

}

• 1

• 2

• 3

• 4

然后执行： npm run build-p

命令行输出没有任何变化，任然是：

process.env.NODE\_ENV 的值是(webpack.config.js)：undefined

但通过浏览器访问/dist/index.html，控制台有如下信息输出：

app test

process.env.NODE\_ENV 的值是(app.js)：production

也就是说，通过webpack -p，然process.env.NODE\_ENV值传递给app.js了（webpack.config.js并未获取到~）

三、通过 webpack.DefinePlugin 定义

接着看，假设webpack.config.js是基本定义，针对上线产品，额外定义了webpack.config.prod.js，然后通过webpack-merge合并两个配置文件webpack.config.prod.js如下：

const webpack = require('webpack')

const merge = require('webpack-merge')

const config = require("./webpack.config.js")

module.exports = merge(config,{

plugins:[

new webpack.DefinePlugin({

'process.env.NODE\_ENV': JSON.stringify('production')

}),

new webpack.optimize.UglifyJsPlugin({

compress: process.env.NODE\_ENV === 'production'

})

]

})

console.log("process.env.NODE\_ENV 的值是(webpack.config.prod.js)："+ process.env.NODE\_ENV)

• 1

• 2

• 3

• 4

• 5

• 6

• 7

• 8

• 9

• 10

• 11

• 12

• 13

• 14

• 15

• 16

可以看到，在这个文件里，我们通过webpack.DefinePlugin定义了process.env.NODE\_ENV，现在在package.json里增加一行，并通过--config指定配置文件：

"scripts": {

"build":"webpack",

"build-p":"webpack -p",

"build-prod":"webpack --config webpack.config.prod.js"

},

• 1

• 2

• 3

• 4

• 5

然后执行：npm run build-prod

命令行里有如下输出：

process.env.NODE\_ENV 的值是(webpack.config.js)：undefined

process.env.NODE\_ENV 的值是(webpack.config.prod.js)：undefined

通过浏览器访问/dist/index.html，控制台有如下信息输出：

app test

process.env.NODE\_ENV 的值是(app.js)：production

这次没有用webpack -p参数，而是在webpack.config.prod.js里通过webpack.DefinePlugin定义了process.env.NODE\_ENV，取得了相当的效果。

四、在config文件里获取NODE\_ENV值

解决了在app.js获取NODE\_ENV的值，如何在webpack配置文件里获取NODE\_ENV的值呢，这样就可以根据不同的值定义相关的参数了，如上所述，答案是：corss-env，在package.json里增加一行：

"scripts": {

"build":"webpack",

"build-p":"webpack -p",

"build-prod":"webpack --config webpack.config.prod.js",

"build-cross-env":"cross-env NODE\_ENV=production webpack"

}

• 1

• 2

• 3

• 4

• 5

• 6

• 7

这里执行：npm run build-cross-env，命令行里会得到：

process.env.NODE\_ENV 的值是(webpack.config.js)：production

通过浏览器访问/dist/index.html，控制台有如下信息输出：

app test

process.env.NODE\_ENV 的值是(app.js)：undefined

可以看到，通过cross-envNODE\_ENV=production，然信息传递给了webpack的配置文件，但app.js并没有获取到。

五、总结

很自然的想到，如果里要在配置文件里和业务代码里，都获取到NODE\_ENV，那将3、4结合起来：

"scripts": {

"build":"webpack",

"build-p":"webpack -p",

"build-prod":"webpack --config webpack.config.prod.js",

"build-cross-env":"cross-env NODE\_ENV=production webpack",

"build-cross-env-with-prod":"cross-env NODE\_ENV=production webpack --config webpack.config.prod.js"

}

• 1

• 2

• 3

• 4

• 5

• 6

• 7

运行： npm run build-cross-env-with-prod，命令行中有显示：

process.env.NODE\_ENV 的值是(webpack.config.js)：production

process.env.NODE\_ENV 的值是(webpack.config.prod.js)：production

通过浏览器访问/dist/index.html，控制台有如下信息输出：

app test

process.env.NODE\_ENV 的值是(app.js)：production