# 一. 安装

webpack4.0以上版本的webpack的脚手架webpack-cli已经被被分离出来了，需要另外安装

npm install webpack -g

npm install webpack-cli -g

# 二.打包

## 1.不使用配置文件打包

### 1.1处理js

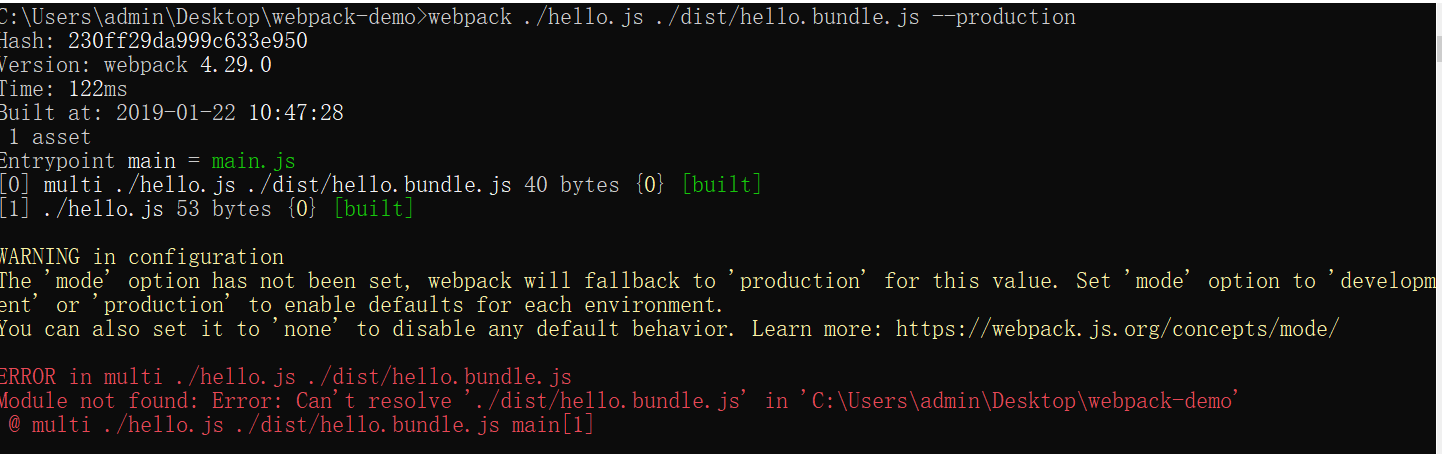
[webpack@4.0](mailto:webpack@4.0)以下版本：webpack hello.js hello.bundle.js

[webpack@4.0](mailto:webpack@4.0)版本及以上：webpack hello.js -o hello.bundle.js (webpack <entry> [<entry>] -o <output>)

还可以安装制定版本：npm install [webpack@3.6.0](mailto:webpack@3.6.0) –save-dev

vue-cli 安装的就是3.6版本

问题解决：添加参数-o



/\*

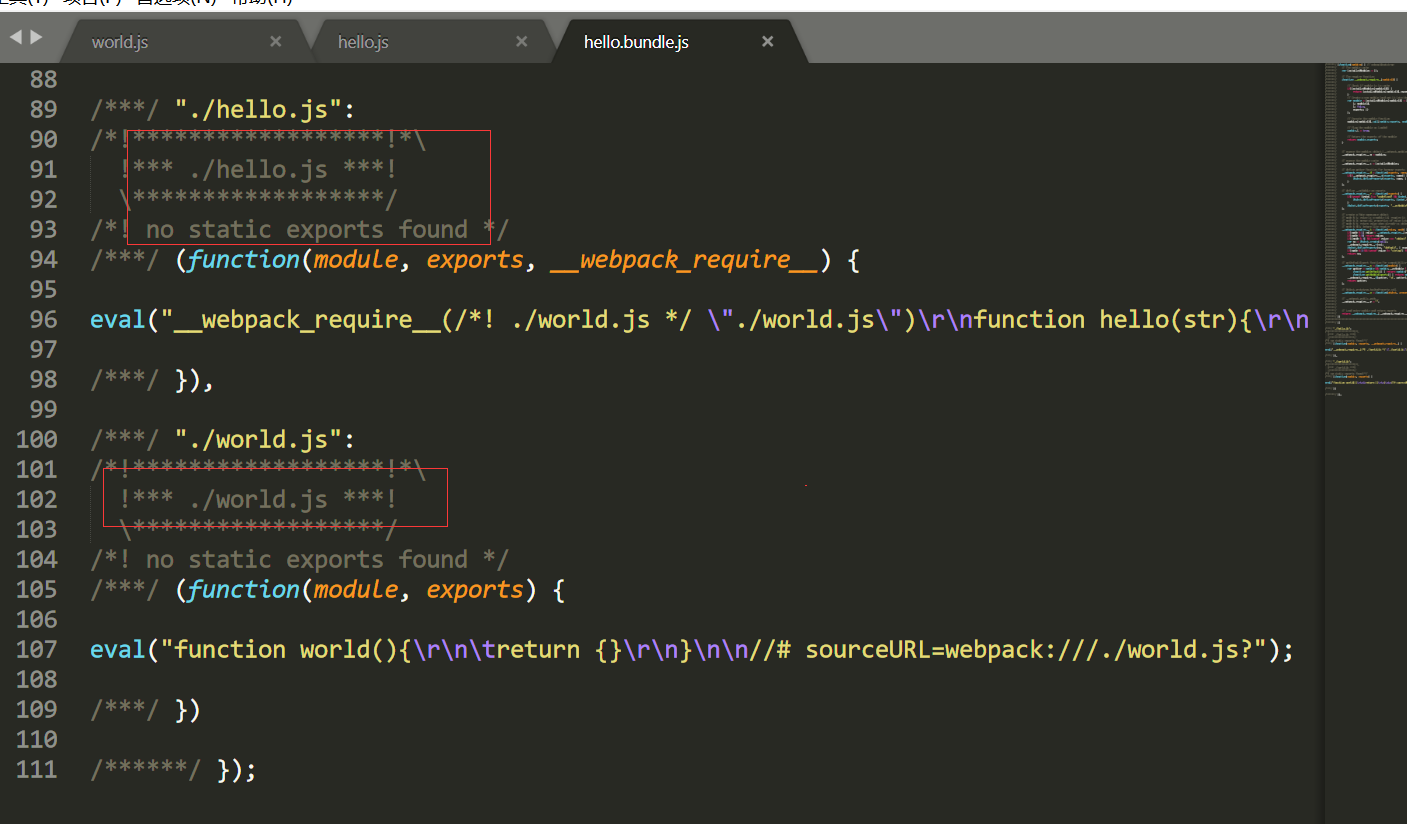
多行注释,不打包新文件

\*/

//单行注释,打包在新文件里

webpack hello.js -output hello.bundle.js --mode=’production’ :指定环境，优先级高于配置文件

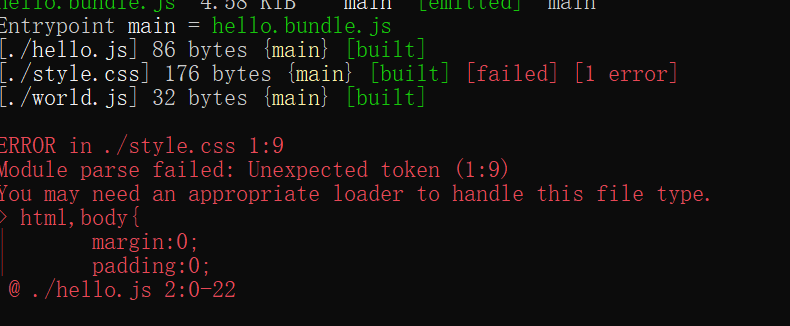
mode设置为development 开发环境打包后的文件是格式化后的



否则就是压缩后的代码

### 1.2处理css

需要依赖loader



1.安装依赖npm install css-loader style-loader --save-dev

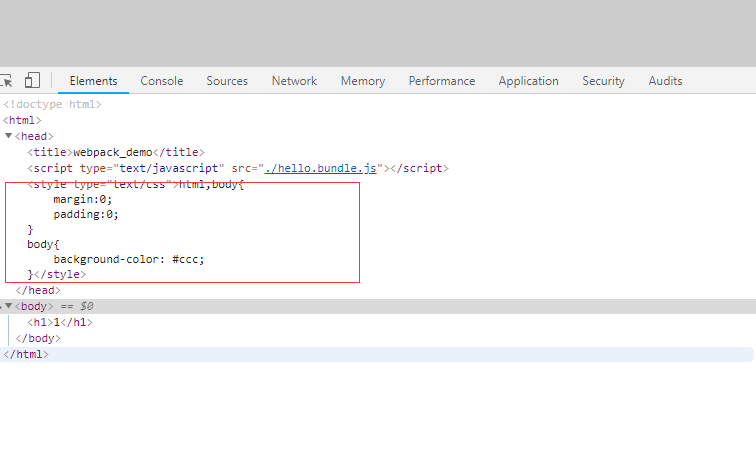
2.指定文件用哪个依赖

require（“css-loader!./style.css”）

3.只引入css-loader 样式不会生效只是可以成功打包，还需要引入style-loader

Require(“style-loader!css-loader!./style.css”)

Style-loader:新建一个<style></style>标签



### 1.3.使用命令行工具指定loader

webpack --help: 列车参数

webpack hello.js -o hello.bundle.js --module-bind “css=style-loader!css-loader”

### 1.4自动编译

1.--watch 参数

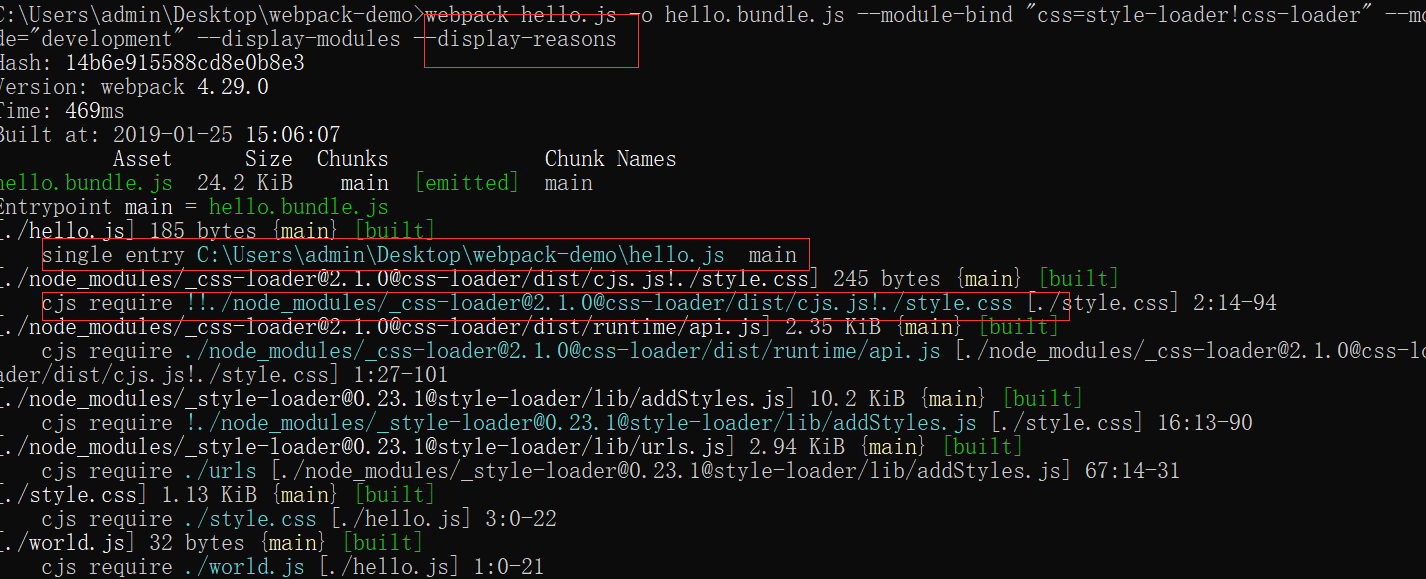
webpack hello.js -o bundle.js –watch: 只要hello.js改变就会被编译

2.--progress: 打包进度

3.--display-modules:列出所有引用的模块



4.--display-reasons: 显示模块包含在输出中的原因



https://webpack.docschina.org/api/cli/#统计数据配置

## 2.配置文件

### 2.1 初始化

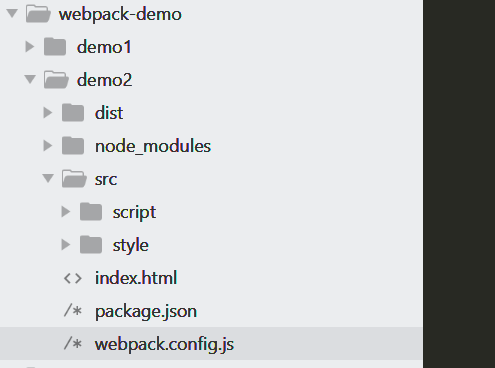
npm init 初始化

npm install webpacl webpack-cli :安装webpack

mkdir src : 存放源文件

mkdir dist:存放打包后的文件

webpack.config.js :新建webpack配置文件



### 2.2 配置文件

1. module.exports={

entry: './src/script/main.js', //入口文件

output:{

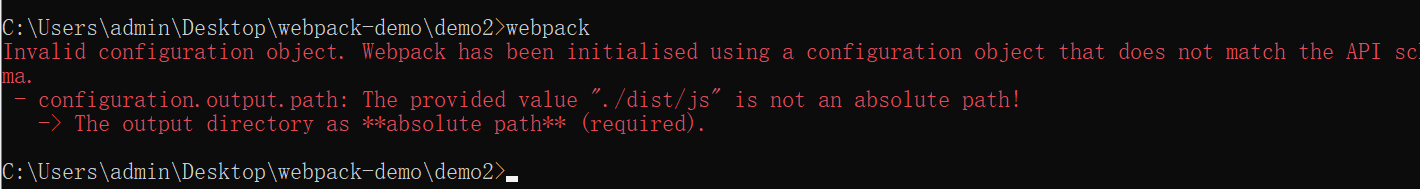
path:'./dist/js',

filename:'bundle.js'

}

}

执行：webpack ,webpack@4版本，输出路径需要写绝对路径



2. 修改output路径为绝对路径



3. webpack 默认采用配置文件是webpack.config.js ,可以利用--cofig参数设置配置文件

webpack --config example.config.js

4. 添加参数 ，利用npm脚本实现



执行：npm run webpack

5. entry

entry: "./app/entry", // string | object | array

entry: ["./app/entry1", "./app/entry2"], //一起打包为一个文件

entry: {

a: "./app/entry-a",

b: ["./app/entry-b1", "./app/entry-b2"]

}

entry值是对象时，输出文件不能是同一个bundle.js；

output的filename: "[name].js", // 用于多个入口点(entry point);

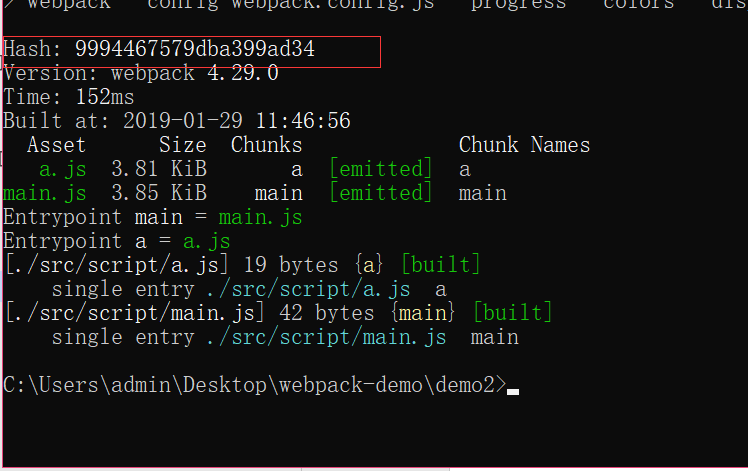
output:{

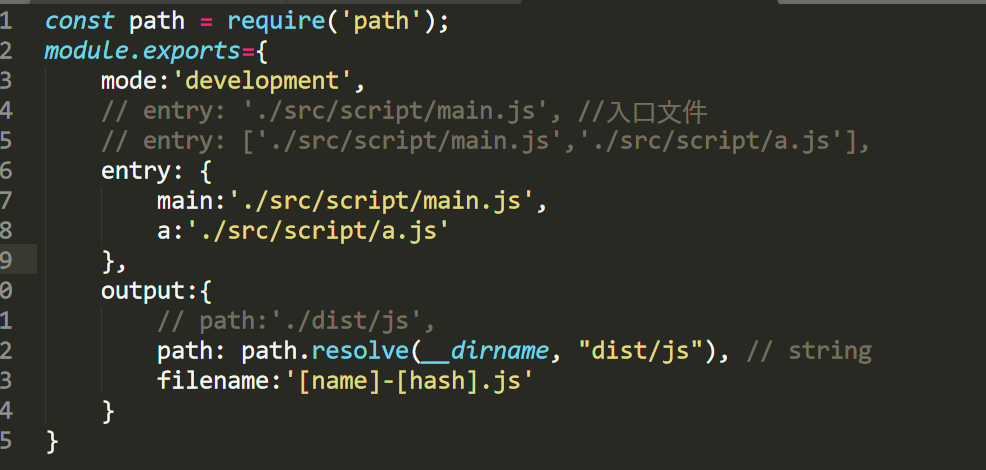
filename: “[chunkhash].js”

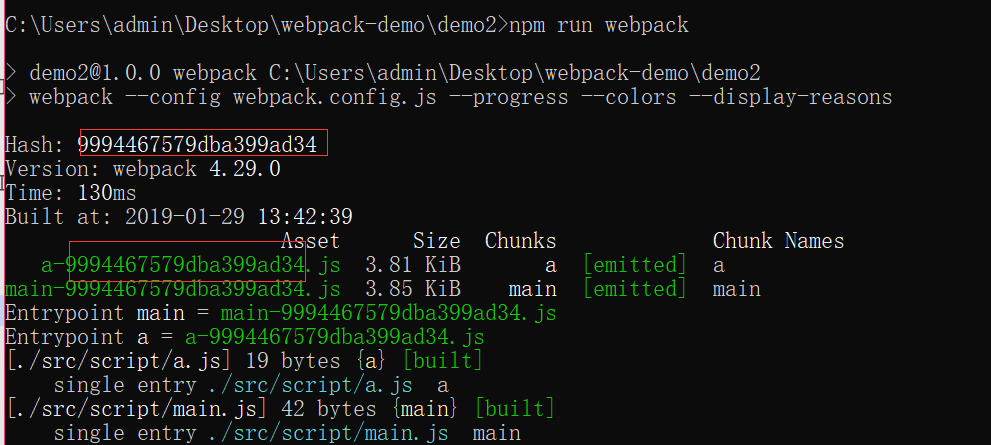
}

filename: “[name]-[hash].js”

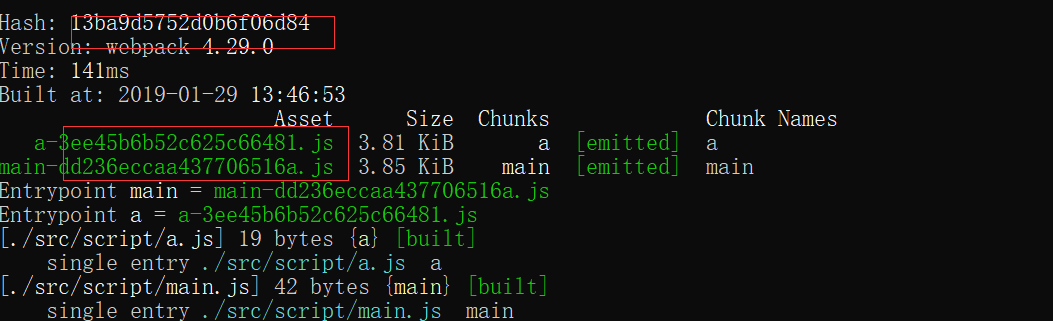
Hash值每次打包生成的







如果是[chunkhash]则每个打包后的hash值都不一样



Hash值可以理解为文件的版本号，也可以认为是文件的md5值，md5值为了保持文件的唯一性，只有文件内容改变hash值才会改变

### 2.3自动化生成项目中的html页面

打包后的文件名是不确定的，避免手动每次更改，自动生成更方便

接用webpack插件：html-webpack-plugin

1.安装

npm install html-webpack-plugin –save-dev

2.使用

在webpack.config.js配置文件里:var htmlWebpackPlugin=require('html-webpack-plugin');

Plugins:[

new HtmlWebpackPlugin()

]



设置一些参数：

plugins: [

new HtmlWebpackPlugin({

template:'index.html',

inject: 'head'，

title:”sdsjd”

})

]

3.设置title:通过参数

Index.html：<title><%= htmlWebpackPlugin.options.title%></title>

plugins: [

new HtmlWebpackPlugin({

filename:’index.html’,

template:'index.html',

inject: 'head'，

title:”title\_demo”,

date: new Date()

})

]

打包后title就是配置文件里的title

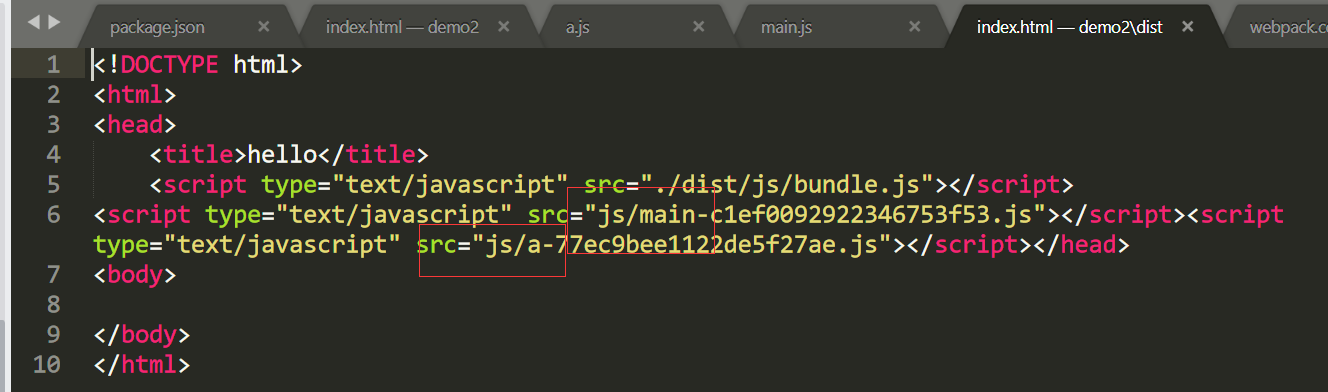
4.publicPath:替换打包后的相对地址，使用线上的绝对地址

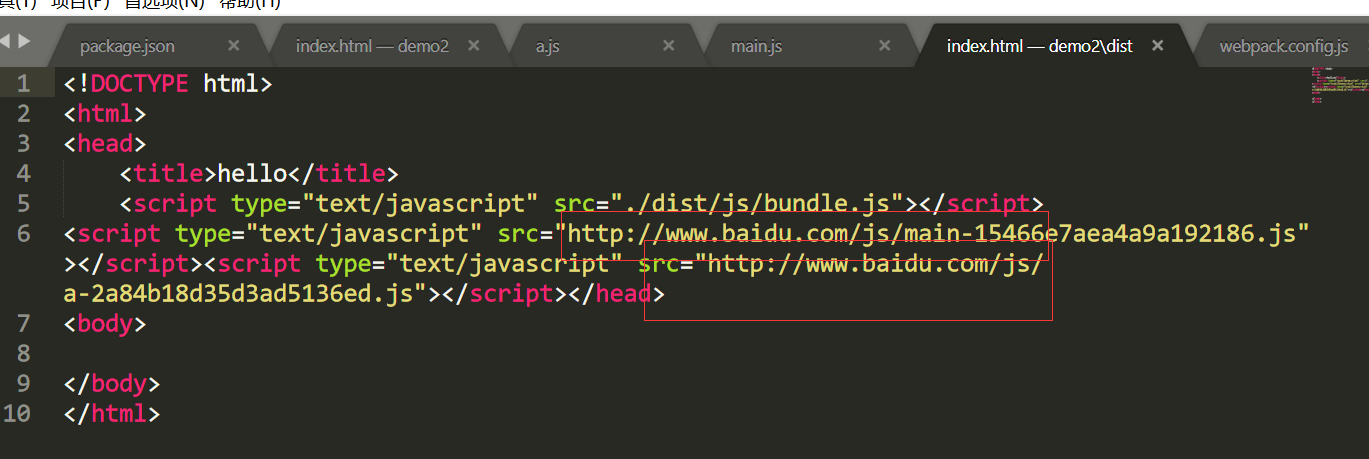
Output :{

path:’./src/script/main.js’,

publicPath: ‘http://\*\*\*\*\*\*/’

}





5.html文件压缩设置

plugins: [

new HtmlWebpackPlugin({

template:'index.html',

inject: 'head',

title:'hello',

minify:{ //压缩

collapseWhitespace:true, // 删除空格

removeComments:true, //删除注释

}

})

]

6.多页面应用

6.1plugins值是一个数组，只需继续添加一个元素即可

plugins: [

new HtmlWebpackPlugin({

template:'index.html', //选择模板文件

inject: 'head',

title:'hello',

minify:{ //压缩html文件

collapseWhitespace:true, // 删除空格

removeComments:true, //删除注释

}

}),

new HtmlWebpackPlugin({

filename:'src/page1.html', // 输出文件

template:'index.html', //选择模板文件

inject: 'head',

minify:{ //压缩html文件

collapseWhitespace:true, // 删除空格

removeComments:true, //删除注释

}

})

]

直接npm run webpack ，每个打包后的html里都会引入所有的js

**6.2 每个页面指引入有用的chunks**

6.2.1参数 chunks:[“a”,”main”] //指引入chunk a 和chunk main;

new HtmlWebpackPlugin({

filename:'src/c.html',

template:'index.html', //选择模板文件

inject: 'head',

title:'this is c.html',

chunks:[“a”,”main”]

})



6.2.2 excludeChunks 引入除了指定的chunk

excludeChunks:[“b”,”c”] //排除b和chunk,引入

**7.循环输出htmlwebpackPLugin 里的内容**

<body>

<!-- 我是注释 -->

<% for(var key in htmlWebpackPlugin) {%>

<%= key %>

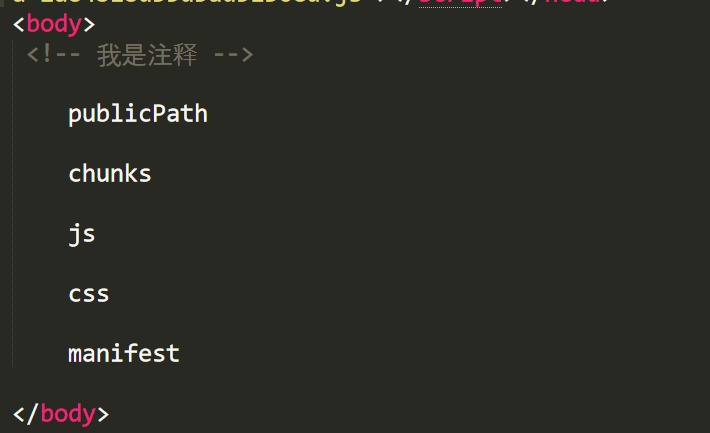
<%}%>

</body>

=>> files

Options

循环htmlWebpackPlugin .files=>>



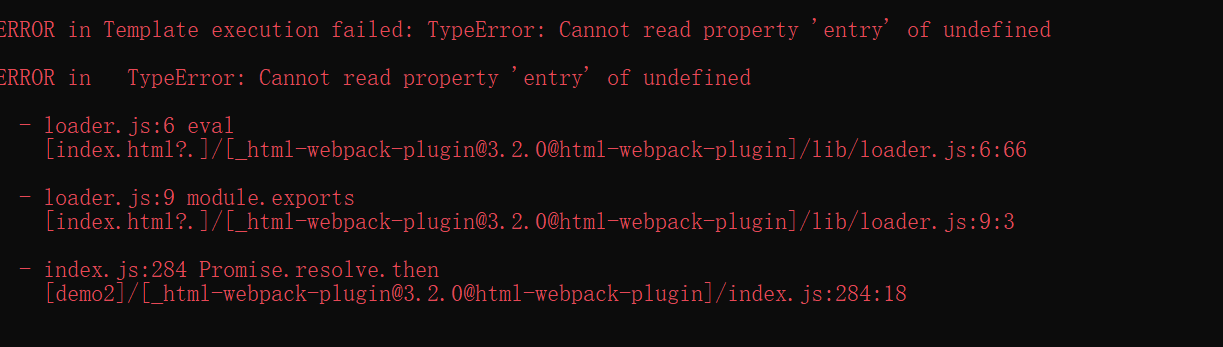
循环htmlWebpackPlugin.files.chunks=>> mian a //根据以下配置的



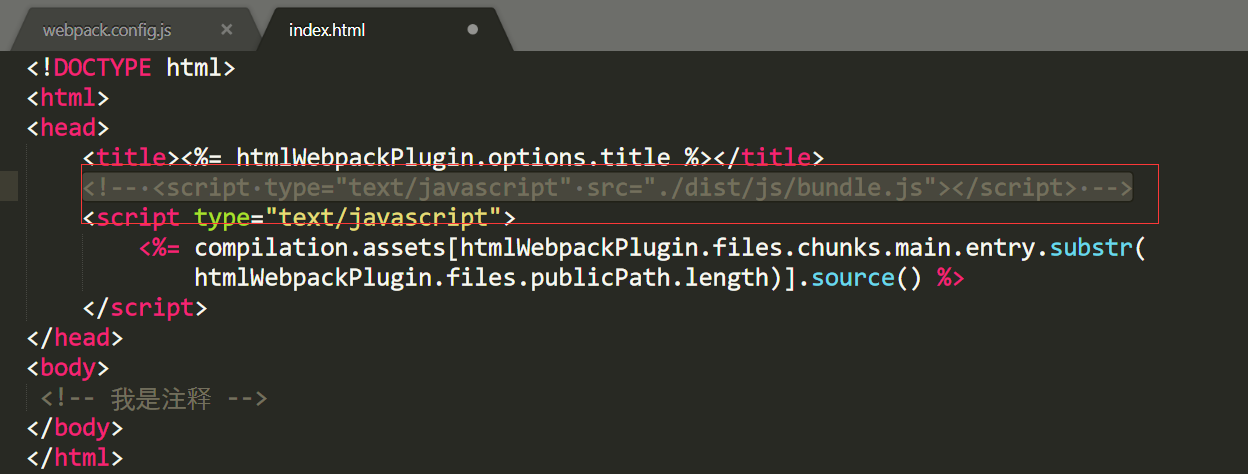
**8.引入文件不以链接的形式嵌入，而是直接渲染 ，减少请求，优化 inline**

<https://github.com/jantimon/html-webpack-plugin/blob/master/examples/inline/template.pug>

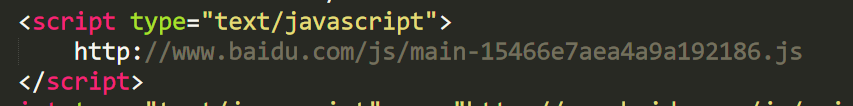
错误提示：



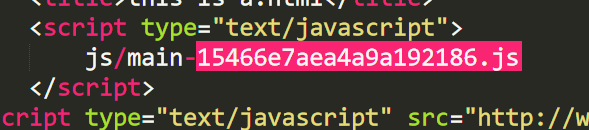
把script标签删掉，注释掉也不行，必须删除



HtmlWebpackPlugin.files.chunks.main.entry //拿到entry完整路径，包含了publicPath



HtmlWebpackPlugin.files.chunks.main.entry.substr(HtmlWebpackPlugin.files.publicPath.length) // 去掉publicPath

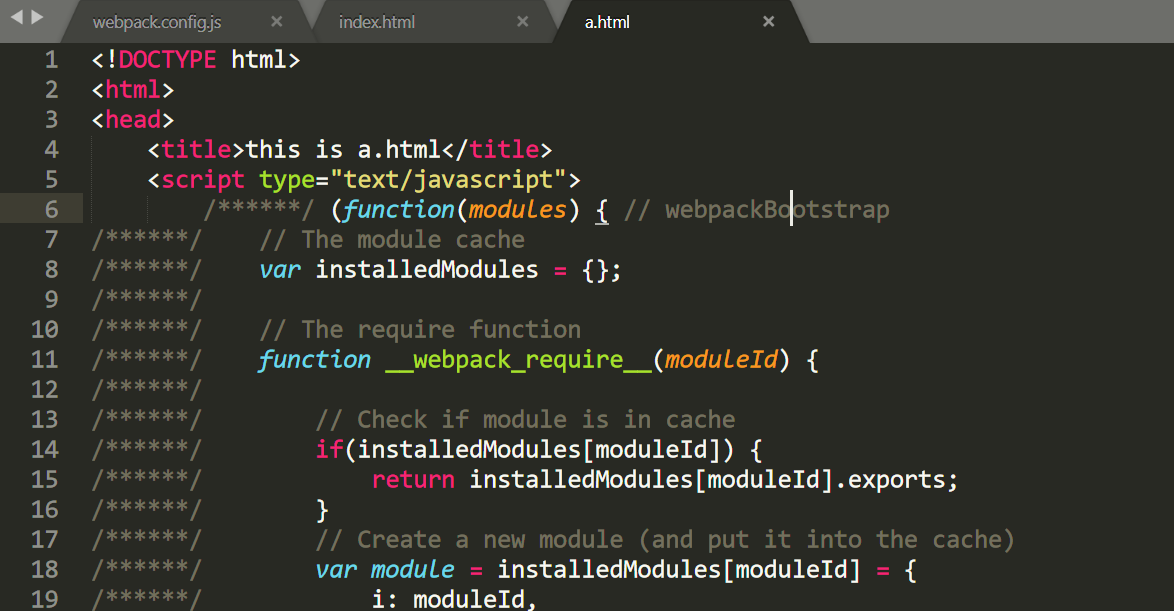


模板文件里定义利用source方法将js内容直接渲染到html文件里

<script type="text/javascript">

<%= **compilation.assets[**htmlWebpackPlugin.files.chunks.main.entry.substr(htmlWebpackPlugin.files.publicPath.length)**].source()** %>

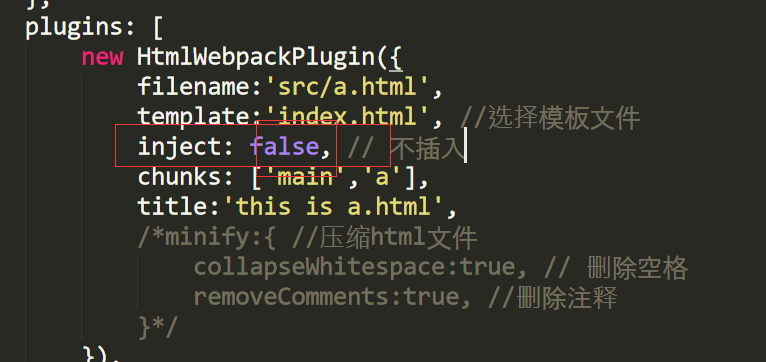
</script>



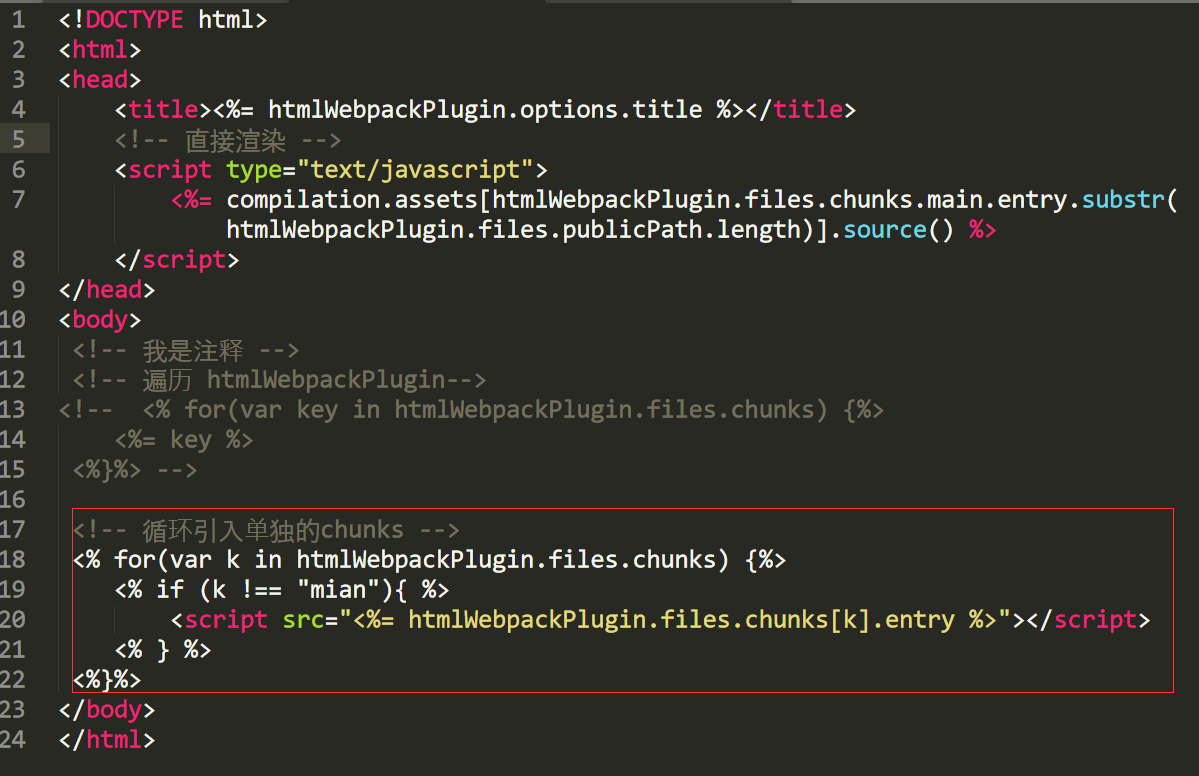
渲染出来的文件除了渲染出内容依旧引入了外部的文件如下图：



解决方法：inject:false



以上统一引入了chunk.mian.entry,可根据场景引入各自的chunks如下图



# 三. 参考文档

1.html-webpack-plugin插件：<https://github.com/jantimon/html-webpack-plugin>

2.webpack官方文档：<https://webpack.docschina.org/configuration/>

3.配置js内容直接渲染在模板文件：

https://github.com/jantimon/html-webpack-plugin/blob/master/examples/inline/template.pug