**webpack手动配置详解**

2020年2月29日

15:04

* **为什么要用webpack**

React的语法是JSX，浏览器并不认识，所以直接引用React写的js文件会报错。

webpack的作用就是将JSX(但不仅限于jsx)编译为浏览器认识的js语法。

* **首先把各个需要手动创建的文件整一下**

计算机生成了可选文字:
〉dist
〉啕nodemodules
全部手动创建
JSindex.js入囗件
0index.htmlht》板
-.babelrc
package-lockjson
0webpack.configjs

**webpack安装**

1. 首先创建webpack项目文件夹例：webpack-base(注:base--基础)
2. vscode打开文件夹终端输入npminit快速初始化一个项目生成package.json文件,装所有运行的包
3. 运行npmiwebpack-cliwebpack--save-dev,安装到项目开发依赖
   * 注意
     + src目录，用于存放项目源代码,常在里面新建入口文件index.js
     + 安装webpack-cli-D这个包的原因是:在webpack4.x以后webpack打包功能由cli这个包来提供
     + **注意:**webpack4.x提供了约定大于配置的概念;目的是为了尽量减少配置文件体积
     + 默认约定了:

打包的入口文件是src->index.js

打包的输出文件是dist->main.js

* 4.x中新增了mode选项(必选项),可选的值为:(1)开发环境:development--注重打包速度(2)生产环境:production--注重打包体积
* **webpack配置文件**
  1. 在项目根目录中创建webpack.config.js并打开加入以下内容

//处理路径的模块  
varpath=require('path');

//导出一个配置对象  
module.exports={

mode:"development",

entry:path.resolve(\_\_dirname,'src/js/index.js'),//项目入口文件,根据自己项目情况写  
output:{  
path:path.resolve(\_\_dirname,'dist'),//输出路径,根据自己项目情况写  
filename:'main.js'//输出文件名,根据自己项目情况写  
}  
}

* **webpack实时打包**
  1. 运行npmiwebpack-dev-server--save-dev安装到开发依赖

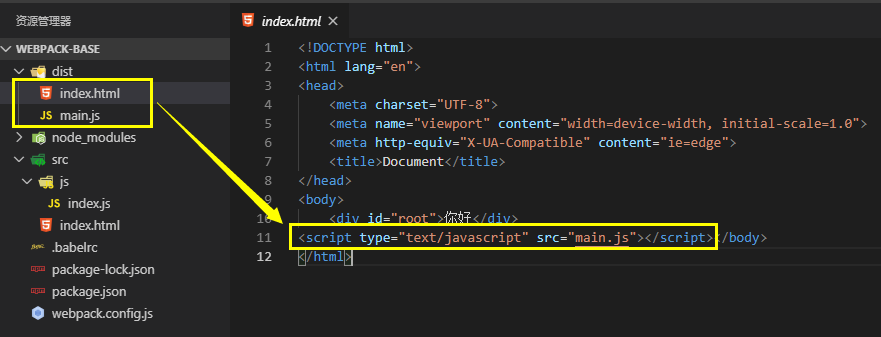
webpack-dev-server作用:帮我们在内存中生成打包好的Js文件

1. 在项目根目录下package.json文件的scripts节点下新增"dev":"webpack-dev-server"指令

* **使用html-webpack-plugin插件配置启动页面**

html-webpack-plugin可以整合webpack编译生成的静态资源,自动生成html文件,

htmlWebpackPlugin作用:帮我们在内存中自动生成首页的插件,并自动把打包好的js文件注入到首页中去



1. 运行npmihtml-webpack-plugin--save-dev安装到开发依赖
2. 修改webpack.config.js配置文件如下:

varpath=require('path');  
//导入插件  
varhtmlWebpackPlugin=require('html-webpack-plugin');

module.exports={

mode:"development",

entry:path.resolve(\_\_dirname,'src/js/index.js'),  
output:{  
path:path.resolve(\_\_dirname,'dist'),  
filename:'main.js'  
},  
plugins:[  
newhtmlWebpackPlugin({  
template:path.resolve(\_\_dirname,'src/index.html'),//模板路径  
filename:'index.html'//生成HTML文件名称  
})  
]  
}

* **Plugin配置拓展**

constHtmlWebpackPlugin=require('html-webpack-plugin');//通过npm安装

npminstallhtml-webpack-pugin–save-dev

Plugins:[

newHtmlWebpackPlugin({

has:true,//向html引入的src链接后面增加一段hash值,消除缓存

chunks:['index'],//添加引入的js,也就是entry中的key

filename:'index.html',//每次调用指定生成html名称

template:\_\_dirname+'/index.html',//模板路径

//minify:{

//collapseWhitespace:true,//折叠空白区,压缩文件

//}

})

]

* **自动打开浏览器、热更新**
  1. 修改package.json的script节点如下,--port3333表示打开的端口号为3333:

"dev":"webpack-dev-server--hot--port3333--open"

* **使用webpack打包css文件**
  1. 运行npmistyle-loadercss-loader--save-dev
  2. 修改webpack.config.js这个配置文件:

module:{//配置第三方loader模块  
rules:[//文件匹配规则  
{test:/\.css$/,use:['style-loader','css-loader']}**(注意:有更改**{test:/\.css$/,loader:'style-loader!css-loader'},)  
]  
}

表示用use规定的loader处理test匹配到的文件;loader模块的调用顺序是从后向前;

* **使用webpack打包less文件**
  1. 运行npmiless-loaderless--save-dev
  2. 修改webpack.config.js这个配置文件：

{test:/\.less$/,use:['style-loader','css-loader','less-loader']},

* **使用webpack打包sass文件**
  1. Npm install --save-dev sass-loader  
     因为sass-loader依赖于node-sass，所以还要安装node-sass  
     npminstall--save-devnode-sass**安装失败解决方案见下文**
  2. 在webpack.config.js中添加处理sass文件的loader模块：

{test:/\.scss$/,use:['style-loader','css-loader','sass-loader']}

* **使用webpack处理css中的路径**
  1. 运行npmiurl-loaderfile-loader--save-dev
  2. 在webpack.config.js中添加处理url路径的loader模块:

file-loader将文件上的import/require()解析为URL并将文件发送到输出目录,经配置可避免重名

url-loader用于将文件转换成base64URIs,一般用于转换小图标和字体图标

{test:/\.(png|jpg|gif)$/,use:'url-loader'},//处理小图标  
{test:/\.(ttf|eot|svg|woff|woff2)$/,use:'url-loader'}//处理字体图标

1. 可以通过limit指定进行base64编码的图片大小;只有小于指定字节(byte)的图片才会进行base64编码:

{test:/\.(png|jpg|gif)$/,use:'url-loader?limit=43960'},

**注意有更改整体更改为**:{test:/\.(gif|jpg|png|woff|svg|eot|ttf)$/,loader:'url-loader?limit=8192&name=images/[hash:8].[name].[ext]'},

* **使用babel处理高级JS语法**
  1. 运行npmibabel-loader@babel/plugin-transform-runtime@babel/core--save-dev安装babel的相关loader包
  2. 运行cnpmi@babel/preset-env--save-dev安装babel转换的语法
  3. 在webpack.config.js中添加相关loader模块,注意一定要把node\_modules文件夹添加到排除项:

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

**注意有更改整体更改为:**

{

test:/\.js$/,

exclude:/(node\_modules|bower\_components)/,

use:{

loader:'babel-loader',//es5转es6

options:{

presets:['@babel/preset-env']

}

}

},

1. 在项目根目录中添加.babelrc文件，并修改这个配置文件如下：

{  
"presets":["@babel/preset-env"],  
"plugins":["@babel/plugin-transform-runtime"]  
}

* **使用babel处理高级HTML语法**
  1. 运行npminstallhtml-withimg-loader–save-dev安装babel转换的语法
  2. 在webpack.config.js中添加处理HTML的loader模块:

{test:/\.(htm|html)$/i,use:['html-withimg-loader']}

* **webpack.config.js文件配置(注意:有部分与上面的配置不同)**

//处理路径的模块

varpath=require('path');

//导入插件

varhtmlWebpackPlugin=require('html-webpack-plugin');

//导出一个配置对象

module.exports={

mode:"development",

entry:path.resolve(\_\_dirname,'src/js/index.js'),//项目入口文件,根据自己项目情况写

output:{

path:path.resolve(\_\_dirname,'dist'),//输出路径,根据自己项目情况写

filename:'main.js'//输出文件名,根据自己项目情况写

},

plugins:[

newhtmlWebpackPlugin({

template:path.resolve(\_\_dirname,'src/index.html'),//模板路径

filename:'index.html'//生成HTML文件名称

})

],

module:{

rules:[

{test:/\.css$/,loader:'style-loader!css-loader'},

{test:/\.less$/,use:['style-loader','css-loader','less-loader']},

{test:/\.(gif|jpg|png|woff|svg|eot|ttf)$/,loader:'url-loader?limit=8192&name=images/[hash:8].[name].[ext]'},

{

test:/\.js$/,

exclude:/(node\_modules|bower\_components)/,

use:{

loader:'babel-loader',//es5转es6

options:{

presets:['@babel/preset-env']

}

}

},

{test:/\.(htm|html)$/i,use:['html-withimg-loader']}

]

}

}

* **npmrundev运行后的显示批注如下:**
* **打包好的文件main.js被托管于内存中,速度快,减少对物理磁盘的伤害**
* 
* **React配置**
  + webpack默认只能打包处理.js后缀名类型的文件;像.png.vue无法主动处理,所以要配置第三方的loader;

注意:如果按照以上配置webpack则在配置react的时候只需要安装下面红色标注的loader

npminstall@babel/core@babel/preset-env@babel/runtime@babel/plugin-transform-runtime@babel/plugin-proposal-class-properties@babel/preset-react-D

这几个包安装一下，然后在.babelrc文件中添加

{"presets":["@babel/preset-env","@babel/preset-react"],"plugins":["@babel/transform-runtime","@babel/plugin-proposal-class-properties"]}

v WEBPACK-BASE 
v dist 
index.html 
JS main.js 
> node_modules 
v js 
orreactEÄ*... 
JS index.js 
g index.html 
.babelrc 
package-lock.json 
package.json 
O webpack.configjs 
"presets" 
"@babel/preset-env", 
"@babel/preset-react" 
"plugins": 
"@babel/transform-runtime" , 
"@babel/plugin-proposal-class-properties" 

v WEBPACK-BASE 
v dist 
index.html 
JS main.js 
> node_modules 
v js 
orreactEÄ*... 
JS index.js 
index.html 
.babelrc 
package-lock.json 
package.json 
O webpack.configjs 
"name" : 
"webpack-base" , 
"version : 
" "I.e.e", 
"description": 
main" • "index.js" 
"scripts" • 
"test": "echo no test specified\" 
"dev": "webpack-dev-server" 
'author": " 
"license : 
"devDependencies 
"@babel/core 
"@babel/preset- 
"@babel/preset 
"@babel/preset-stage-e 
&& exit 1", 
"A7.8.6", 
"@babel/plugin-proposal-class-properties 
"@babel/plugin-transform-runtime ' 
'A7.8.3", 
env": "A7.8.6", 
"A7.8.3", 
-react 
"A7.8.3", 
"A7.8.3", 
"@babel/runtime 
"A7.8.4••, 
"babel- loader 
" css-loader ' 
" file-loader 
"A8.e.6", 
• A3.4.2", 
"As.l.e", 
"html-webpack-plugin": "AB 2 

v js 
orreactEÄ*... 
JS index.js 
index.html 
.babelrc 
package-lock.json 
package.json 
O webpack.configjs 
/\.css$/, 
loader: 'style-loader!css-loader' 
test : 
/\.less$/, use: [ 'style-loader' , 
•css-loader', 'less-loader'] 
test : 
pnglwofflsvgl eotlttf)$/, loader• 
. url 
test : 
loader: 'babel- loader' , //esS#es6 
optio 
presets: [ '@babel/preset-env' ] 
test : 
(html html)$/i, 
use : 
[ 'html -withimg-loader ' ] 

* **如何省略.jsx后缀**

webpack.config.js

resolve:{

extensions:['.js','.jsx','.json']//表示，这几个文件的后缀名,可以省略不写

}

* **使用@符号代替相对路径**

alias:{//表示别名

'@':path.join(\_\_dirname,'./src')//这样,@就表示项目根目录中src的这一层路径

}

* **为普通样式表,通过modules启用模块化**

大家可以在css-loader之后,通过?追加参数

其中,有个固定的参数,叫做modules,表示为普通的css样式表,启用模块化

{test:/\.css$/,loader:['style-loader','css-loader?modules']},//打包处理CSS样式表的第三方loader

* **使用localIdentName来自定义模块化的类名**
  + 解决React中css-loader中的optionshasanunknownproperty'localIdentName'.Thesepropertiesarevalid报错问题
  + css-loader更新后，获取localIdentName不再是在后面跟参数获取，而是进行分开
  + 更新前：{test:/\.css$/,use:['style-loader','css-loader?modules&localIdentName=[path][name]-[local]-[hash:5]']},
  + 更新后：{test:/\.css$/,use:['style-loader',{loader:'css-loader',options:{modules:{localIdentName:'[path]'},}}]}
* **在项目中启用模块化并同时使用bootstrap**
  + npmibootstrap@3.3.7-S
    - 在组件中引入import'../../node\_modules/bootstrap/dist/css/bootstrap.css'
    - woff2参数不要忘记
* 为了使用sass，我们需要安装sass的依赖包

在项目下，运行下列命令行  
npminstall--save-devsass-loader  
因为sass-loader依赖于node-sass，所以还要安装node-sass  
npminstall--save-devnode-sass

* {test:/\.scss$/,use:['style-loader',{loader:'css-loader',options:{modules:{localIdentName:'[path][name]-[local]-[hash:5]'},}},'sass-loader']},//打包处理scss文件的loader
* **node-sass安装失败的各种坑**
  + 第一种解决方案

最后gypERR!builderror应该是没有安装什么

解决办法：

主要是windows平台缺少编译环境，

1、先运行：npminstall-gnode-gyp

2、然后运行：运行npminstall--global--productionwindows-build-tools可以自动安装跨平台的编译器

npminstallnode-sass-D

报错

解决办法：

删除之前安装失败的包(第一次安装请跳过此步)

npmuninstallnode-sass

重新安装

npminstallnode-sass-D

* 第二种解决方案
  + 以管理员身份运行cmd打开桌面上的python-2.7.10.amd64.msi安装包(别忘了勾选最后的ADD)
  + 安装完后,cmd输入python检查是否安装成功
  + 配置环境变量可知乎查询
    - 计算机生成了可选文字:
      辑用户变里
      变單名QD
      系單〔s〕
      CLASSPÅTH
      ComSpe
      FPNOHOST0
      hadoop„
      stall
      PATH
      〔a已d丑@
      ．亡证yh“7
      《Jar.%JÅVÅ0
      胃ind“、sy三t“32、cm土
      、m，-pr囗“、hadoop一乙3
      蕪辑〔1〕
      册〔L〕
* npmuninstallnode-sass
* 重启计算机

重新安装

npminstallnode-sass-D

* 把自己的样式表,定义为.scss文件
* 第三方的样式表,还是以.css结尾
* 我们只需要为自己的.scss文件,启用模块化即可
* 运行npmisass-loadernode-sass-D安装能够解析scss文件的loader
* **Development和Production的区别**
* mode有两个可选值:production不支持监听,侧重于打包后的文件大小,development侧重于构建的速度

Development开发环境

侧重于构建的速度考虑的是方便开发和调试,不需要考虑代码合并和样式分离这些。

Production生产环境

侧重于打包后的文件大小生产环境打包是要最后发布到服务器部署的代码,需要保持代码简洁,加载性能最优，不需要调试辅助工具。不支持监听

* **npmi安装命令中的-g-D-S的区别**
  1. npmixx-g：npminstallxx--global的简写，对模块进行全局安装。
     1. 所谓全局安装，是指把模块安装到操作系统上，全局是指操作系统，全局安装完成后，一般会安装到AppDataAppData\Roaming\npm目录下。如：npminstallwebpack-g，就是全局安装webpack，在操作系统的任何一个目录下都可以使用webpack所提供的指令。
  2. npmixx-D：npminstallxx--save-dev的简写，对模块进行局部安装，模块写入到devDependencies对象。
     1. 局部的意思是只针对当前项目，模块一般安装到项目文件夹下的node\_modules文件夹下。
     2. devDependencies对象，是我们开发的时候需要用到的一些包，只用于开发阶段，真正打包上线的时候并不需要这些包，因为这些工具只是你用来打包代码的，是用来识别特定文件以及代码，帮助我们生产最终文件的。如npmivue-loadervue-template-complier-D,就是在Vue项目中安装vue模板文件的解析插件，经过配置后即可在项目中解析vue模板
  3. npmixx-S：npminstallxx--save的简写，同上也是对模块进行局部安装，不同的是模块写入到dependencies对象。
     1. 模块同上一样将安装到项目文件夹下的node\_modules文件夹下。
     2. dependencies对象，这个与devDependencies不同，是需要发布到生产环境中的，就比如你要跑一个基于vue的项目，所以需要vue.js来支持，vue.js文件就需要跟随项目到最终的生产环境。npmivue-S即可将Vue模块安装到项目的依赖中，并一同发布到生产环境。

1. 下图为项目根目录下的package.json中的devDependencies和dependencies对象



配置webpack终极版命令

2020年2月29日

16:05

* **需提前要手动创建的文件**

计算机生成了可选文字:
〉dist
〉啕nodemodules
全部手动创建
JSindex.js入囗件
0index.htmlht》板
-.babelrc
package-lockjson
0webpack.configjs

* **所有终端指令**
  1. npminit

1. npmiwebpack-cliwebpack-D
2. npmiwebpack-dev-server-D
3. npmihtml-webpack-plugin-D
4. npmistyle-loadercss-loader-D
5. npmiless-loaderless-D
6. npminstall--save-devsass-loader  
   因为sass-loader依赖于node-sass，所以还要安装node-sass  
   npminstall--save-devnode-sass
7. npmiurl-loaderfile-loader-D
8. npmibabel-loader@babel/plugin-transform-runtime@babel/core-D
9. npmi@babel/preset-env-D
10. npminstallhtml-withimg-loader-D
11. webpack
12. npmrundev调用的是Package.json中"dev":"webpack-dev-server--hot--port3333--open"

* **.babelrc文件配置**

计算机生成了可选文字:
〉dist
〉啕nodemodules
JSindex.js
0index.html
.babelrc
package-lock.jsc》n
package.json
0webpack.configjs
presets
"@babel/preset-env"
"plugins
"@babel/plugin-transform-runtime"

{

"presets":[

"@babel/preset-env"

],

"plugins":[

"@babel/plugin-transform-runtime"

]

}

* **Package.json配置增加dev指令属性**

计算机生成了可选文字:
c炻onx
name
Vet'SIon
〉嘔dist
〉啕nodemodules
JSindex.js
0index.html
-.babelrc
package-lockjson
packagejson
0webpack.configjs
"webpack-base"
"description'
maln
scripts
"test"
"author"
"index.js"
"echo\"Error:notestSecified\"&&exit1"
-hot--port彐3彐彐--open"
“ISC"
"devDependencies，
"@babel/core"
。^7．8．6“
"@babel/plugin
—transform—runtime
"@babel/preset
》^7．8．6“
—env
"babel一loader'
"file-loader"
"html-webpack-plugin"
"html-withimg-loader"
。^彐．11．1到
"less-loader"
"style-loader'
"url-loader"
"webpack"
》^4．41．5“
^3．1．16"
"webpack-cli"
。^彐．彐．11"
"webpack-dev-server"
^3．13．彐"

"dev":"webpack-dev-server--hot--port3333--open"

* **webpack.config.js文件配置**

//处理路径的模块

varpath=require('path');

//导入插件

varhtmlWebpackPlugin=require('html-webpack-plugin');

//导出一个配置对象

module.exports={

mode:"development",

entry:path.resolve(\_\_dirname,'src/js/index.js'),//项目入口文件,根据自己项目情况写

output:{

path:path.resolve(\_\_dirname,'dist'),//输出路径,根据自己项目情况写

filename:'main.js'//输出文件名,根据自己项目情况写

},

plugins:[

newhtmlWebpackPlugin({

template:path.resolve(\_\_dirname,'src/index.html'),//模板路径

filename:'index.html'//生成HTML文件名称

})

],

module:{

rules:[

{test:/\.css$/,loader:'style-loader!css-loader'},

{test:/\.less$/,use:['style-loader','css-loader','less-loader']},

{test:/\.(gif|jpg|png|woff|woff2|svg|eot|ttf)$/,loader:'url-loader?limit=8192&name=images/[hash:8].[name].[ext]'},

{

test:/\.js$/,

exclude:/(node\_modules|bower\_components)/,

use:{

loader:'babel-loader',//es5转es6

options:{

presets:['@babel/preset-env']

}

}

},

{test:/\.(htm|html)$/i,use:['html-withimg-loader']}

]

}

}