**webpack + git+node**

**前端学习--利用webpack4手把手带你搭建一个vue多页面应用**

转自知乎用户-----[**灰蓝宇墨**](https://www.zhihu.com/people/hui-lan-yu-mo)

利用webpack4一步步搭建一个vue的开发环境，会详细讲解webpack4中的知识点。本文篇幅比较长，建议跟着文章一起敲一遍代码，你会发现webpack4也不过如此。

所有的源码可以在github上获取

[ddhujiang/webpack4-vue​github.com](https://link.zhihu.com/?target=https%3A//github.com/ddhujiang/webpack4-vue)

**前端所有学习资源，戳下面链接**

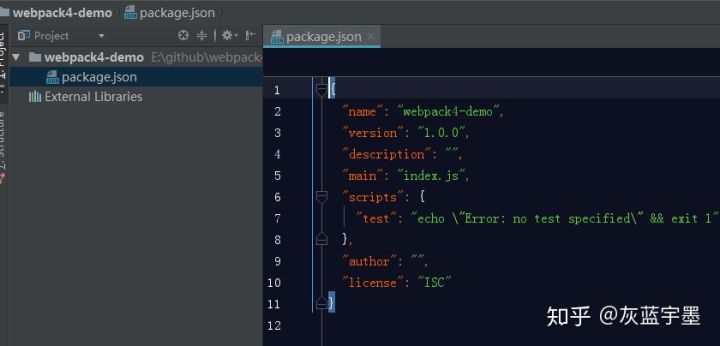
[灰蓝宇墨：前端看这里​zhuanlan.zhihu.com](https://zhuanlan.zhihu.com/p/146653449)

**准备工作**

新建文件夹webpack4-vue,初始化生成package.json文件,命令行会有一些小提示，一路回车就好了。

cnpm init //初始化生成package.json文件

package.json，就是包管理的json文件，初始化长这样



**解析：**

npm

* 允许用户从NPM服务器下载别人编写的第三方包到本地使用。

cnpm

* 淘宝团队做的国内镜像，因为npm的服务器位于国外可能会影响安装。淘宝镜像与官方同步频率目前为 10分钟 一次以保证尽量与官方服务同步。
* 安装：命令提示符执行npm install cnpm -g registry=https://registry.npm.taobao.org
* cnpm -v 来测试是否成功安装

npm \ cnpm功能都一样,但一般我们都是用cnpm,因为下载更快。

**安装webpack、webpack-cli**

**webpack4必须安装webpack-cli,可以全局安装，但我还是建议跟着项目走，那样不同的项目可以安装不同的版本。**

cnpm i webpack webpack-cli -D

**解析：**

i 即install 的缩写

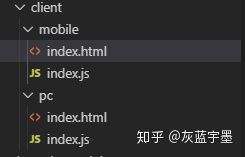
-g -S -D

* -g：即--global全局安装。 将会安装在C：\ Users \ Administrator \ AppData \ Roaming \ npm，并且写入系统环境变量；
* -S：即 --save，写入package.json的dependencies ,dependencies 是需要发布到生产环境的
* -D：即--save-dev，写入package.json的devDependencies ,devDependencies 里面的插件只用于开发环境，不用于生产环境。

本文所有的插件、loader等安装都是通过这种格式，接下来所有需要安装我就不再一一列举了

**多入口打包**

新建如下文件



配置webpack.config.js文件

const path = require('path');

module.exports = {

// 入口文件

entry: {

pc: path.join(\_\_dirname, "/client/pc/index.js"),

mobile: path.join(\_\_dirname, "/client/mobile/index.js")

},

//打包输出文件

output: {

path: path.join(\_\_dirname, "/dist"), //打包后的文件存放的地方

filename: "js/[name]-[hash:6].js", //打包后输出文件的文件名

}

}

**解析：**

**\_\_dirname**

\_\_dirname`是node.js中的一个全局变量，它指向当前执行脚本所在的目录E:\github\webpack4-demo(这是我的目录)

**path.join**

path是node中自带的模块，通过const path = require('path');

path.join(path1，path2，path3.......)

作用：将路径片段使用特定的分隔符（window：\）连接起来形成路径，并规范化生成的路径。若任意一个路径片段类型错误，会报错。

**filename**

webpack会为每个生成的Chunk取一个名称，Chunk的名称和entry的配置有关：

1. 如果entry是一个string或者array，就只会生成一个chunk，这个chunk的名称是main;
2. 如果entry是一个object，就可能出现多个chunk，这时chunk的名称是object键值对里键的名称

然而，当通过多个入口起点(entry point)、代码拆分(code splitting)或各种插件(plugin)创建多个 bundle，应该使用以下一种替换方式，来赋予每个 bundle 一个唯一的名称……使用入口名称：

output:{

filename: "[name].bundle.js"

}

使用内部 chunk id

output:{

filename: "[id].bundle.js"

}

使用每次构建过程中，唯一的 hash 生成

output:{

filename: "[name].[hash].bundle.js"

}

使用基于每个 chunk 内容的 hash：

output:{

filename: "[chunkhash].bundle.js"

}

上面介绍的 id、name、hash、chunkhash等都是webpack内置变量，

id是唯一标示，不会重复，从0开始，

name 是模块名称，是你自己起的，在配置路由懒加载的时候可以自己命名

**webpack.dev.js**

**我们的项目会运行在开发和生产两个环境，一般我们都会把webpack的配置分开。**

新建webpack.dev.js用于开发环境打包，新建webpack.prod.js用于生产环境打包

webpack.dev.js

**const** merge **=** require('webpack-merge'); *// 引入webpack-merge功能模块*

**const** common **=** require('../webpack.config.js'); *// 引入webpack.config.js*

module.exports **=** merge(common, { *// 将webpack.config.js合并到当前文件*

devServer**:** {

contentBase**:** "./dist", *// 本地服务器所加载文件的目录*

port**:** "8018", *// 设置端口号为8018*

inline**:** **true**, *// 文件修改后实时刷新*

historyApiFallback**:** **true**, *//不跳转*

hot**:** **true** , *// 热更新*

proxy**:** {

'/'**:** {

target**:** 'http://localhost:8018', *// 你项目的本地服务地址*

bypass**:** **function**(req, res, proxyOptions) {

**const** userAgent **=** req.headers['user-agent'];

**if** (req.headers.accept.indexOf('html') **!==** **-**1) {

*// 根据访问终端返回模板*

**if** (/mobile/i.test(userAgent) **&&** **!**/iPad/i.test(userAgent)) {

**return** '/index.mobile.html';

}

**return** '/index.pc.html';

}

},

},

},

},

plugins**:** [

]

});

**解析：**

webpack-merge用于合并webpack的配置文件

**webpack-dev-server**

devServer是用来提高开发效率的，不是用devServer来做打包的，它提供了一些配置项，可以用于改变devServer的默认行为，要配置devServer，除了可以在配置文件里通过devServer传入参数，还可以通过命令行传入参数。

注意！！！只有在通过webpack-dev-server启动webpack时，配置文件里的devServer才会生效，因为这些参数所对应的功能都是webpack-dev-server提供的，webpack本身并不认识devServer的配置项。

这样我们在本地打包完之后就能直接看到运行的效果了devServer作为webpack配置选项中的一项，以下是它的一些配置选项:

contentBase

设置服务器所读取文件的目录，当前我们设置为"./dist"

port

这个没什么说的了就是端口号，默认是8080，如果占用就换成8081或者1234，都可以

open

open 在devServer启动且第一次构建完成时，自动用我们的系统的默认浏览器打开网页

proxy

代理，解决开发环境的跨域问题

mmodule.exports **=** {

devServer**:** {

proxy**:** {

'/api'**:** 'http://localhost:3000'

}

}

};

请求到 /api/xxx 现在会被代理到请求 http://localhost:3000/api/xxx, 例如 /api/user 现在会被代理到请求 http://localhost:3000/api/user

本文是用它根据不同的终端返回不同的首页页面。

**webpack.prod.js**

生产环境

**const** merge **=** require('webpack-merge');

**const** common **=** require('../webpack.config.js');

**const** {CleanWebpackPlugin} **=** require('clean-webpack-plugin'); *// 引入CleanWebpackPlugin插件*

​

module.exports **=** merge(common, { *// 将webpack.common.js合并到当前文件*

plugins**:** [

**new** CleanWebpackPlugin(), *// 所要清理的文件夹名称*

]

})

CleanWebpackPlugin，每次打包前，清除上次打包生成的文件

**package.json**

**我们可以在这个文件中编辑我们运行项目的命令**

"scripts": {

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

"dev": "cross-env NODE\_ENV=development webpack-dev-server --open --config config/webpack.dev.js --mode development"

},

**解析:**

当使用NODE\_ENV =production来设置环境变量时，根据平台(windows、linux等)不一样会有一些差异,cross-env使得您可以使用单个命令，而不必担心为平台正确设置或使用环境变量

//cross-env 一样是需要安装的

cnpm install cross-env --save-dev

--mode

mode/--mode参数来表示是开发还是生产环境

**现在：**

cnpm run dev 可以运行并预览项目

cnpm run build 打包项目并发布上线

**vue**

现在我们已经实现了简单的打包，但对于项目开发肯定是不够的。我们还需要引入js的框架，对css也需要进行打包。

**vue-loader**

安装vue-loader

报错

vue-loader was used without the corresponding plugin. Make sure to include VueLoaderPlugin in your webpack config.

是因为Vue-loader在15.\*之后的版本都是 vue-loader的使用都是需要伴生 VueLoaderPlugin的,

const VueLoaderPlugin = require('vue-loader/lib/plugin');

plugins: [

new VueLoaderPlugin(),

]

**css-loade**

在项目中写入一些 css内容，需要css-loader vue-style-loader 进行处理

{

//解析器的执行顺序是从下往上(先css-loader再vue-style-loader)

test: /\.css$/,

use: [

{

loader: MinCssExtractPlugin.loader,// 将处理后的CSS代码提取为独立的CSS文件

},

'vue-style-loader',

'css-loader'

],

},

Loaders需要单独安装并且需要在webpack.config.js中的modules配置项下进行配置，Loaders的配置包括以下几方面：

* test：一个用以匹配loaders所处理文件的拓展名的正则表达式（必须）
* loader：loader的名称（必须）
* include/exclude：手动添加必须处理的文件（文件夹）或屏蔽不需要处理的文件（文件夹）（可选）；
* options：为loaders提供额外的设置选项（可选）

**less**

直接写css会比较麻烦，一般我们都会使用一门css的扩展语言

安装less-loader less

大家千万不要忘记安装less

这时css和js是打包在一起的，如何进行css分离

**提取css**

报错，ReferenceError: document is not defined

因为style-loader和 mini-css-extract-plugin 冲突，去除掉style-loader即可(style-loader和vue-style-loader功能一样，.vue文件用vue-style-loader)

**const** MinCssExtractPlugin **=** require( "mini-css-extract-plugin" ); *// 将css代码提取为独立文件的插件*

module**:**{

rules**:**[{

test**:** /\.vue$/,

loader**:** 'vue-loader',

},

{

*//解析器的执行顺序是从下往上(先css-loader再vue-style-loader)*

test**:** /\.css$/,

use**:** [

{

loader**:** MinCssExtractPlugin.loader,*// 将处理后的CSS代码提取为独立的CSS文件*

},

'css-loader'

],

},

{

test**:** /\.less$/,

use**:** [

{

loader**:** MinCssExtractPlugin.loader,

},

{

loader**:**'css-loader'

},

{

loader**:**'less-loader'

}

],

},

]

},

**使用babel转码**

ES6的语法大多数浏览器依旧不支持,bable可以把ES6转码成ES5语法，这样我们就可以大胆的在项目中使用最新特性了

cnpm install babel-core babel-loader babel-preset-env babel-preset-stage-3 --save-dev

在项目根目录新建一个.babelrc文件

{

"presets": [

["env", { "modules": false }],

"stage-3"

]

}

webpack.config.js添加一个loader

{

test**:** /\.js$/,

loader**:** 'babel-loader',

exclude**:** /node\_modules/

}

*//exclude表示忽略node\_modules文件夹下的文件，不用转码*

报错

在配置webpack.config.js自动打包的时候,出现Error: Cannot find module '@babel/core'错误最初以为是babel-core没有安装上。重装了好几遍babel-core还是不行。对照以前的项目,发现babel-loader的版本不一样,之前的是@7.1.5版本,而现在是@8.0.0版本。

回退babel-loader的版本

npm uninstall babel-loader --save-dev

npm install babel-loader@7.1.5 --save-dev

**官方默认babel-loader | babel 对应的版本需要一致: 即babel-loader需要搭配最新版本babel**

两种解决方案:

1. 回退低版本

npm install -D babel-loader@7 babel-core babel-preset-env

1. 更新到最高版本:

npm install -D babel-loader @babel/core @babel/preset-env webpack

***三更灯火五更鸡，正是男儿读书时*。**

**❤️关注+点赞+收藏+评论+转发❤️**，原创不易，鼓励笔者创作更好的文章

关注公众号***前端码头***，获取独家学习路线+精品课程，更多前端小干货等着你喔