### Webpack小纪

△--save和--save-dev的区别：

1. --save会把依赖包名称添加到package.json文件dependencies中
2. --save-dev添加到package.json文件devDependencies中

△“dependencies”： 这些包都是你的应用程序在生产环境中所需要的。

“devDepedencies”：这些包只是在开发和测试中需要的。

**1、**在更高层面，在 webpack 的配置中 loader 有两个目标：test 属性，用于标识出应该被 对应的 loader 进行转换的某个或某些文件。

use 属性，表示进行转换时，应该使用哪个 loader。

关于test和use？？？

module: {

rules: [

{ test: /\.txt$/, use: 'raw-loader' }

]

}

**2、**webpack是一个打包工具，通过input进去，通过output输出。webpack只认识js结尾 的文件，通过loader把文件转换成以js为结尾的文件。plugin用来解决loader未能解 决的问题。loader相当于一个预处理，plugin把代码进行压缩减少代码体积，辅助开发。

1. webpak使用loader处理文件，使用loader打包除.js结尾的文件。使用loader让webpack 识别文件。
2. 各种loader的下载方法和作用：

（1）url-loader：当图片大小小于limit限制是，会把图片打包成64格式的图片；当大

于的时候，会把图片打包成单独的图片。若图片小，则用url-loader，打包到代码 中。

（2）fill-loader：打包图片

（3）url-loader：也用来打包图片，但是可以设置limit限制，把图片打包到代码中

（4）style-loader：把cssloader生成的内容插入到header中

（5）css-loader：处理css文件 （先执行）

（6）sass-loader：处理sass文件

（7）node-sass：把sass文件转换成css文件 【因网络环境问题，下载node-sass时出 现问题。

解决方法：设置变量 sass\_binary\_site，指向淘宝镜像地址。示例：npm config set sass\_binary\_site https://npm.taobao.org/mirrors/node-sass/

然后npm install一下，然后在正常下载npm i node-sass -D

（8）postcss-loader:给类似于reansform的属性添加前缀，如webkit-transform等，并且 在根目录下创建postcss.config.js。

（9）autoprefixer：帮助我们自动添加前缀

autoprefixer使用出现问题，在postcss.config.js中使用autoprefixer并不能自动添加：

module.exports = {

plugins: [

require('autoprefixer')

]

}

解决方法：在package.json中添加以下代码（针对各个浏览器去做处理）

"browserslist": [

"defaults",

"not ie < 11",

"last 2 versions",

"> 1%",

"iOS 7",

"last 3 iOS versions"

]

1. plugin：在我们打包的某一时刻做一些事情
2. 各种plugin的下载方法和作用：

（1）HtmlWebpackPlugin：在dist文件夹中自动生成html文件，命令行：npm i html-webpack-plugin -D

（2）CleanWebpackPlugin：清空dist文件夹，再进行打包 命令行： npm i

CleanWebpackPlugin -D

（3）在package.json中的scripts中配置：“build”： “webpack --watch”在控制台运行 npm run build，会实时保存文件的修改状态，但是需要手动刷新浏览器。

在package.json中的scripts中配置：“server”：“webpack-dev-server”在终端下载 npm i webpack-dev-serve -D,并在webpack.config.js中配置devServer:{contentBase: './dist'}.实时监测文件的修改状态，自动重新打包，并在浏览器中自动更新。

再配置一个open：true。每次npm run server的时候，会帮我们自动打开浏览器

配置proxy: {“/api”: “http://localhost:3000”}，访问这个地方，实现proxy的跨域代理， vue和react都能用proxy的原因是其脚手架都使用了webpack-dev-server

（4）HotModuleReplacementPlugin:浏览器实行热更新 并在webpack.congif.js中引用 webpack。在devServer中配置hot:true;hotOnly: true;（不会自动刷新页面）

注意：css不需要手动去更改

Js：手动更改；判断是否有module.hot。在index.js中

if(module.hot) {

  module.hot.accept('./number2', function() {

    document.body.removeChild(document.getElementById('number2'));

    number2();//这部分是代码示例

  })

}

1. ES6转换ES5：babel：npm install --save-dev babel-loader babel-loader @babel/core

Babel-loader:帮助webpack打包的工具

@babel.core:是babel的核心库，能够帮助识别代码中的内容

@babel/preser-env:ES6转换成ES5

@babel/polyfill：防止低版本浏览器不支持so下载。缺点：把所有的内容都加载打包

“presets”：[[‘@babel/preser-env,{useBuildIns: ‘usage’}]]

userBuiltIns:按需加载，根据使用填充代码，解决业务代码用。会生成全局变量，污染 环境

Transform-runtime：若有第三方库，不会污染全局变量。修改“core.js”：false（不填充） 改成2（支持全局变量和静态属性）改成3（支持全局变量和静态属性，还有一些实例 的属性）。

1. SourceMap（打包速度慢）：解决实际代码和源代码不一致时，通过sourcemap去debug 到源代码的技术，现在大多数代码都是通过打包压缩，通过这个来定位到源代码中 的一些错误
   1. 在webpack.config.js的module.exoprts中的mode下面配置devtool： ‘source-map’，这样的话源代码的错误会映射出来，在js文件的多少行都会有显示
   2. inline-source-map会被打包到main.js中，能够映射到第三方模块的错误 || source-map会被打包成一个外部文件 || inline-cheap-source-map只关注行，速度会更快，第三方模块的错误不会处理 || eval

△可以组合使用

△开发环境推荐cheap-module-eval-souce-map

生产环境推荐cheap-module-souce-map

1. 分离Development和Production环境

通过创建webpack.common.js（公共的）、webpack.dev.js（开发环境）webpack-prod.js（生产环境）三个文件分离生产环境和开发环境，并且下载一个npm i webpack-merge -D去合并这三个文件

1. 打包分离css文件并压缩合并

分离：（1）下载插件：npm i mini-css-extract-plugin -D

（2）插件引入到页面中：const .....

（3）在plugins中new插件

（4）在rules中把style-loader替换成MiniCssExtractPlugin.loader

压缩合并：（1）下载OptimizeCSSAssetsPlugin插件

（2）插件引入到页面中：const .....

（3）在OptimizeCSSAssetsPlugin插件中new插件

11、webpack实现对React框架代码的打包

△window系统一定要配置“linebreak-style”: 0，不然就会出现error Expected linebreaks to be 'LF' but found 'CRLF' 报错

（1）下载react-dom 和react插件

（2）新建babelrc文件

 {

   "presets": [

     ["@babel/preset-env",

      {

        useBuiltIns: 'usage'

      }

    ],

      "@babel/preset-react"

  ]

}

1. webpack的eslint的校验配置
2. 安装npm i eslint -D
3. 初始化npx eslint --init -->生成。Ecliptic’s
4. 配置eslint : npm i babel-ealint -D
5. 用webpack配置eslint: npm i eslint-loader -D
6. webpack优化
7. 更新技术迭代Node、Webpack、Npm
8. Webpack优化体积

①生产环境使用压缩功能，包括js、css/图片、svg等（mode：production是压缩模式）

②设置合理的SPlitChunks分组（合理的划分资源的大小，提高缓存命中率，从而降低 资源的加载时间）

③使用lodash、momentjs这类库，需按需引入

④合理使用polyfill，防止多余代码

1. 遇到.jsx结尾的文件会报错，所以在webpack.common.js中配置：resolve：{extensions：[‘.js’, ‘.jsx’]}

①使用resolve.extensions优先查找

②resolve.alias减少查找过程

③排除不需要解析的模块noParse

④合理配置