webpack安装和使用

webpack的理解：

功能：

webpack是基于node，把项目开发过程中的模块文件按照不同的类型进行打包

原理：

webpack的打包是依赖模块化的开发方式，所以，我们项目是使用了模块化的开发 - 把模块化开发中的一个一个的单独的模块文件，根据其依赖关系进行查找，然后打包。 - 打包后的文件通过他内置的es5的模块化方案（require.js）实现了一个小巧的模块化加载器

webpack安装过程：

初始化，生成package.json文件

>**npm init**

安装webpack

>**npm install --save-dev webpack**

或

>npm install --D webpack

注意：可以全局，但是不推荐：因为不同的项目可能使用了不同版本的webpack，推荐不同的项目使用不同的版本，所以webpack最好是安装在项目目录下

根据提示安装CLI或安装command

>**npm i webpack-cli --D**

或

>npm i webpack-command --D

新建src和dist文件夹

src是原始文件，dis是转义后的文件

注意：package.json文件里面的项目名称和安装文件的名称不一样

webpack打包方式：

方法一：不需要配置文件命令行

>.\node\_modles\.bin\webpack .\src\a.js -o .\dist\bundle.js ——生成环境

>.\node\_modules\.bin\webpack .\src\a.js -o .\dist\bundle.js --mode=development———开发环境

>**npx webpack .\src\a.js --out-fire .\dist\bundle.js --mode=development**

方法二：配置文件的命令

**生成webpack.config.js**

在文件中：

module.exprots={

mode:'development'——开发环境

entry: .\src.js——入口文件

output: {——出口文件

path: \_\_dirname+'dist',————必须绝对路径

filename: 'my-first-webpack.bundle.js'

}

或者

推荐：

const path = require('path');

module.exprots={

mode:'development'

entry: .\src.js

output: {

path: path.resolve(\_\_dirname, 'dist'),————必须绝对路径

filename: 'my-first-webpack.bundle.js'

}

执行命令：

>**npx webpack**

>**npx webpack -w**——实时编译

webpack.config.js配置文件的理解：

**mode**: 'development'：设置打包的时候为开发模式

**entry**：入口文件

形式：

单入口文件：'./src/index.js'

多入口文件：['./src/index.js', './src/list.js']

**context**：配置项目通用目录，对entry，loader加载的路径进行配置，他的值必须是一个绝对路径，默认是当前文件所在目录的绝对路径‘

**output**：打包后的文件存放的目录，注意：path必须是一个绝对路径

虽然直接使用 \_\_dirname 就可以了，但是推荐我们使用 path.resolve方法来处理一下路径文件：因为不同操作系统所使用文件系统不一样

形式：

path: \_\_dirname+'dist',————必须绝对路径

path: path.resolve(\_\_dirname, 'dist'),————必须绝对路径

filename的形式：比如，[] 是这里的一个特殊标记，表示里面的名称是一个占位符，类似变量的作用，[]中的值是约定好的，具体看文档

libraryTarget：’umd‘：默认用require.js的后置模块cmd，现在用umd形式来加载模块。



webpack的执行流程：

流程：读取(entry) -> 打包 -> 输出(output)

中间件：运行在读取以后，输出之前，中间件能够获取到webpack读取到文件内容，同时会返回中间件处理的结果给webpack进行输出

中间件又分成两种不同模式：loader,plugin

loader：运行在打包之前

plugin：运行在打包之后

最终流程：读取 -> loader -> 打包 -> plugin -> 输出

loader的理解：

webpack默认只能处理js文件，如果我们希望webpack帮助我们同时打包项目中其他类型文件，比如txt\图片\css，就需要使用loader来处理;

步骤：

loader 需要自己单独安装

在webpack.config.js的module中进行的配置

注意：use同时加载两个文件，有先后顺序，后面的那个先执行，再执行前面的

'style-loader'：将模块的导出作为样式添加到 DOM 中

'css-loader'： 解析 CSS 文件后，使用 import 加载，并且返回 CSS 代码

形式：

module: {

rules: **[**

**// 配置文件处理的规则**

**{**

**// test是正则，用来描述处理的文件名规则**

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

**// 指定使用什么loader来处理test匹配到文件**

**use: 'raw-loader'**

**}**

**]**

}



plugin的理解：

功能：打包优化压缩等

步骤：

plugin需要自己单独安装

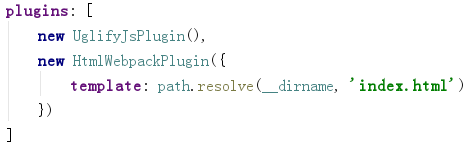
头部用require引入文件夹

在webpack.config.js的plugins中进行的配置

UglifyJsPlugin：压缩文件

HtmlWebpackPlugin：HTML文件中不用script引入js入口文件，只要安装这个插件，webpack可以自动引入

clipboard.png



vue-cli : vue脚手架

全局安装：

>**npm install -g @vue/cli** 或者 yarn add global @vue/cli

>**vue -h**——查看vue脚手架的使用方法

>**vue -v**——查看当前vue脚手架的版本

项目：

创建项目：

>**vue create 项目名称**

或

>**vue init webpack 项目名称**

启动项目:

>**cd 项目名称**

>**npm run dev**——写在package.json里面script里面的命令

全局安装：

clipboard.png

创建项目：

clipboard.png

启动命令：

