1. 注意：
   1. 所有构建工具都是基于nodejs平台运行的，模块化默认采用commonjs
2. 功能：
   1. Webpack为一个静态模块打包器，会将前端的所有资源（js/json/css/img/less/...）都作为模块处理，再进行打包
   2. 能处理js/json资源，不能处理css/img等资源
   3. 能将ES6+模块化编译为浏览器能识别的模块化
3. 核心概念：入口（Entry）【入口文件】、输出（Output）【输出文件】、Loader【用于处理非js文件】、插件（Plugins）、模式（Mode）
   1. 模式：development、production。生产环境模式会进行代码压缩
4. 使用：
   1. 运行指令：webpack 入口文件路径 -o 输出文件所在文件夹路径 --mode=development / production
   2. 配置开发环境：创建配置文件：项目根目录创建webpack.config.js，使用commonjs
   3. module.exports=｛属性：值，...｝
   4. entry：‘入口路径’，output：｛xx｝，module：｛配置对象｝，plugins：[详细plugins的配置][，resolve：｛｝]，mode：‘xx’
   5. output配置：

filename：‘输出文件名’

path：‘输出文件所在文件夹的绝对路径’

environment：{ arrowFunction：false } 不使用箭头函数

* 1. module配置：loader需要通过npm下载

rules：[｛loader配置对象｝, ...]

loader配置对象：

test：/\.css$/ 正则表达式，表示匹配哪些文件

exclude：/\.css$/ 或 /node-modules/ 正则表达式，表示排除哪些文件

use：[‘loader1’，‘loader2’...] 使用指定loader，loader从后往前执行

loader：‘loader’ 当只使用一个loader时，不使用use，而用该属性

option：｛配置对象｝ 可选

limit：8\*1024 图片小于8KB时，就会被base64处理（减少请求数，图片更大）

esModule：false 关闭url-loader的es6模块，解决[object Module]问题

name：‘[hash：10].[ext]’ 重命名文件，取哈希值前10位加上原来的扩展名

outputPath：‘imgs’ 以输出文件夹为起点，设置输出文件夹

引入不同的文件：将文件或文件夹放入项目根目录下的src文件夹中

css：需把文件引入入口文件

test：/\.css$/

use：[‘style-loader’， 创建style标签，将js中的样式资源插入，添加到head中生效

‘css-loader’] 可以将css文件作为commonjs模块加载js中，然后打包

less：需首先下载less模块，并把文件引入入口文件

test：/\.less$/

use：['style-loader','css-loader','less-loader'] less-loader可以将less文件转换为css文件

在css中被引用的图片：需下载url-loader和file-loader。新版本可以自动打包

test：/\.（jpg | png | gif）$

loader：‘url-loader’，options：｛limit：8\*1024，name：‘[hash：10].[ext]’，esModule：false，outputPath: 'imgs'｝

在html中被引用的img图片：负责引入img，从而能被url-loder处理

test：/\.html$/

loader：‘html-loader’

其他资源：无需压缩的文件，如字体等

exclude：/\.（css | js | html | less）$/

loader：’file-loader’

options：｛name：‘[hash：10].[ext]’｝

ts文件：

test：/\.ts$/

loader：’ts-loader’

exclude：/node-modules/

项目根目录创建tsconfig.json，使用es6，项目中的package.json的“script”中添加属性“build”：“webpack”

使用命令npm run build构建

* 1. plugins配置：

html：首先通过npm安装html-webpack-plugin插件，该插件会默认创建一个空的html，自动引入打包输出的所有资源

在配置文件开头const HTMLWebpackPlugin = require（‘html-webpack-plugin’）；

在plugins数组内new HTMLWebpackPlugin（）

需要结构：new xx（可选对象）

属性：

template：‘被复制的html文件路径’ 按照模板生产html文件

title：“标题” html的标题

devServer：热加载（自动编译，自动打开浏览器，自动刷新浏览器），只会在内存中编译打包，不会有输出

下载webpack-dev-server包，在项目的package.json的“scripts”中添加属性"start": "webpack serve --open"

启动devServer的指令为：npm start

clean-webpack-plugin：每次编译时清除上次输出的文件

配置文件开头：const {CleanWebpackPlugin} = require('clean-webpack-plugin')

plugins数组内：new CleanWebpackPlugin（）

* 1. resolve配置：

extensions：[‘.扩展名’] 指定扩展名可以用作引用模块，新版本不需要

* 1. 配置生产环境：
     1. 提取css成单独文件：安装mini-css-extract-plugin
        1. rules中：
           1. test: /\.css$/,
           2. use: [MiniCssExtractPlugin.loader, 'css-loader'] css-loader可以将css文件整合到js文件中
        2. plugins中添加new MiniCssExtractPlugin({filename: 'css/built.css' }) 可命名输出的文件
     2. css兼容性问题：安装postcss、postcss-loader、postcss-preset-env
        1. 项目的package.json中添加：
           1. "browserslist": { // 开发环境 --> 设置node环境变量：process.env.NODE\_ENV = development

"development": ["last 1 chrome version","last 1 firefox version","last 1 safari version" ],

"production": [">0.2%","not dead","not op\_mini all"] } 生产环境：默认是看生产环境

* + - 1. 上述提取css成单独文件的use数组内添加一个｛｝，加在css-loader之后，less-loader之后，属性如下：
         1. loader：‘postcss-loader’，
         2. options：｛postcssOptions：｛plugins：[“postcss-preset-env”]｝｝
    1. 压缩css：安装optimize-css-assets-webpack-plugin
       1. plugins中添加new OptimizeCssAssetsWebpackPlugin()
    2. js的语法检查：安装eslint-loader、eslint
       1. 设置语法规则：package.json中添加：
          1. "eslintConfig": {"extends": "airbnb-base"} 需要安装eslint-config-airbnb-base、eslint-plugin-import、eslint
       2. rules中添加｛｝，属性如下：
          1. test: /\.js$/,
          2. exclude: /node\_modules/, 排除第三方库的文件
          3. loader: 'eslint-loader',
          4. options: { fix: true } 自动修复eslint的错误
       3. 忽略下一行的eslint检查（js中）：在代码上一行添加注释// eslint-disable-next-line
    3. ts兼容性处理：babel-loader、@babel/core、@babel/preset-env、core-js
       1. test: /\.ts$/，exclude: /node\_modules/,
       2. use:[｛babel配置｝,‘ts-loader’] 若要转换js文件，去掉ts-loader即可
          1. loader：“babel-loader”，
          2. options: {

presets: [ ['@babel/preset-env', 指定环境插件

{ useBuiltIns: 'usage', corejs: {version: 3}, 指定corejs的版本和它的使用方法（usage：按需加载）

targets: {chrome: '60', firefox: '60', ie: '9', safari: '10', edge: '17'} ] ] }