webpack工程化实战

webpack5.x已经在今年10.1国庆期间发布上线，但是考虑到不稳定性，和生态插件的更新，所以本课程仍以webpack4.x版本为主。(webpack@4.43.0，webpack-cli@3.3.12，webpack-dev-server:@3.11.0)

webpack入门

1.webpack简介

Webpack 是一个现代 JavaScript 应用程序的*静态模块打包器*（*module bundler*），当 webpack 处理应用程序时，它会递归地构建一个*依赖关系图(dependency graph)*，其中包含应用程序需要的每个模块，然后将所有这些模块打包成一个或多个 *bundle*。

`Webpack是一个打包模块化JavaScript的工具，它会从入口模块出发，识别出源码中的模块化导入语句，递归地找出入口文件的所有依赖，将入口和其所有的依赖打包到一个单独的文件中`

是工程化、自动化思想在前端开发中的体现。

2.webpack 安装

环境准备

nodeJs： https://nodejs.org/en/

版本参考官网发布的最新版本，可以提升webpack的打包速度

安装方式

局部安装（推荐）

```
npm init -y  # 初始化npm配置文件
npm install --save-dev webpack # 安装核心库
npm install --save-dev webpack-cli # 安装命令行工具


# 安装最新的4.x稳定版本
npm i -D webpack@4.44.0

# 安装指定版本
npm i -D webpack@<version>
```

全局安装（不推荐）

```
# 安装webpack V4+版本时，需要额外安装webpack-cli
npm install webpack webpack-cli -g

# 检查版本
webpack -v

# 卸载
npm uninstall webpack webpack-cli -g
```

`全局安装webpack，这会将你项目中的webpack锁定到指定版本，造成不同的项目中因为webpack依赖不同版本而导致冲突，构建失败`

3.启动webpack

3.1- webpack默认配置

- webpack默认支持JS模块和JSON模块
- 支持CommonJS Es moudule AMD等模块类型
- webpack4支持零配置使用,但是很弱，稍微复杂些的场景都需要额外扩展

3.2- 准备执行构建

- 新建src文件夹
- 新建src/index.js、src/index.json、src/other.js

```
### index.js
const json = require("./index.json");//commonJS
import { add } from "./other.js";//es module
console.log(json, add(2, 3));

### index.json
{
  "name": "JOSN"
}

### other.js
export function add(n1, n2) {
  return n1 + n2;
}
```

3.3- 执行构建

```
# npx方式
npx webpack

# npm script
npm run test
```

修改package.json文件：

```
"scripts": {
  "test": "webpack"
},
```

`原理就是通过shell脚本在node_modules/.bin目录下创建一个软链接。`

3.4-构建成功

我们会发现目录下多出一个`dist`目录，里面有个`main.js`，这个文件是一个可执行的JavaScript文件，里面包含webpackBootstrap启动函数。

3.5-默认配置

```
const path = require("path");
module.exports = {
  // 必填  webpack执行构建入口
  entry: "./src/index.js",
  output: {
    //  将所有依赖的模块合并输出到main.js
    filename: "main.js",
    //  输出文件的存放路径，必须是绝对路径
    path: path.resolve(__dirname, "./dist")
  }
};
```

4.webpack配置核心概念

- chunk：指代码块，一个 chunk 可能由多个模块组合而成，也用于代码合并与分割。
- bundle：资源经过Webpack 流程解析编译后最终结输出的成果文件。
- entry：顾名思义，就是入口起点，用来告诉webpack用哪个文件作为构建依赖图的起点。webpack会根据entry递归的去寻找依赖，每个依赖都将被它处理，最后输出到打包成果中。
- output：output配置描述了webpack打包的输出配置，包含输出文件的命名、位置等信息。
- loader：默认情况下，webpack仅支持`.js .json`文件，通过loader，可以让它解析其他类型的文件，充当翻译官的角色。理论上只要有相应的loader，就可以处理任何类型的文件。
- plugin：loader主要的职责是让webpack认识更多的文件类型，而plugin的职责则是让其可以控制构建流程，从而执行一些特殊的任务。插件的功能非常强大，可以完成各种各样的任务。
  - webpack的功能补充
- mode：4.0开始，webpack支持零配置，旨在为开发人员减少上手难度，同时加入了mode的概念，用于指定打包的目标环境，以便在打包的过程中启用webpack针对不同的环境下内置的优化。。

暗号：你好，webpack

作业：

总结webpack核心概念，做成成思维导图

截图的方式发上来

零配置

零配置是很弱的，特定的需求，总是需要自己进行配置

webpack有默认的配置文件，叫`webpack.config.js`，我们可以对这个文件进行修改，进行个性化配置

- 使用默认的配置文件：webpack.config.js
- 不使用自定义配置文件： 比如webpackconfig.js，可以通过--config webpackconfig.js来指定webpack使用哪个配置文件来执行构建

webpack.config.js配置基础结构

```
module.exports = {
  entry: "./src/index.js", //打包入口文件
  output: "./dist", //输出结构
  mode: "production", //打包环境
  module: {
    rules: [
      //loader模块处理
      {
        test: /\.css$/,
        use: "style-loader"
      }
    ]
  },
  plugins: [new HtmlWebpackPlugin()] //插件配置
};
```

4.1-entry:

指定webpack打包入口文件:Webpack 执行构建的第一步将从 Entry 开始，可抽象成输入

```
//单入口 SPA，本质是个字符串
entry:{
  main: './src/index.js'
}
```

```
==相当于简写===
entry:"./src/index.js"


//多入口 entry是个对象
entry:{
  index:"./src/index.js",
  login:"./src/login.js"
}
```

4.2-output:

打包转换后的文件输出到磁盘位置:输出结果，在 Webpack 经过一系列处理并得出最终想要的代码后输出结果。

```
output: {
    filename: "bundle.js",//输出文件的名称
    path: path.resolve(__dirname, "dist")//输出文件到磁盘的目录，必须是绝对路径
},

//多入口的处理
output: {
    filename: "[name][chunkhash:8].js",//利用占位符，文件名称不要重复
    path: path.resolve(__dirname, "dist")//输出文件到磁盘的目录，必须是绝对路径
},
```

4.3-mode

Mode用来指定当前的构建环境

- production
- development
- none

设置mode可以自动触发webpack内置的函数，达到优化的效果

开发阶段的开启会有利于热更新的处理，识别哪个模块变化

生产阶段的开启会有帮助模块压缩，处理副作用等一些功能

4.4-loader

模块解析，模块转换器，用于把模块原内容按照需求转换成新内容。

webpack是模块打包工具，而模块不仅仅是js，还可以是css，图片或者其他格式

但是webpack默认只知道如何处理**js和JSON**模块，那么其他格式的模块处理，和处理方式就需要loader了

常见的loader

```
style-loader
css-loader
less-loader
sass-loader
ts-loader //将Ts转换成js
babel-loader//转换ES6、7等js新特性语法
file-loader//处理图片子图
eslint-loader
...
```

4.5-moudle

模块，在 Webpack 里一切皆模块，一个模块对应着一个文件。Webpack 会从配置的 Entry 开始递归找出所有依赖的模块。

当webpack处理到不认识的模块时，需要在webpack中的module处进行配置，当检测到是什么格式的模块，使用什么loader来处理。

```
module:{
  rules:[
  {
    test:/\.xxx$/,//指定匹配规则
    use:{
      loader: 'xxx-load'//指定使用的loader
    }
  }
  ]
}
```

案例：样式处理：

css-loader 分析css模块之间的关系，并合成一个css

Style-loader 会把css-loader生成的内容，以style挂载到页面的heade部分

```
npm install style-loader css-loader -D
```

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

5.Plugins：webpack的扩展补充

- 作用于webpack打包整个过程
- webpack的打包过程是有（生命周期概念）钩子

plugin 可以在webpack运行到某个阶段的时候，帮你做一些事情，类似于生命周期的概念

扩展插件，在 Webpack 构建流程中的特定时机注入扩展逻辑来改变构建结果或做你想要的事情。

作用于整个构建过程

HtmlWebpackPlugin

htmlwebpackplugin会在打包结束后，自动生成一个html文件，并把打包生成的js模块引入到该html中。

```
npm install --save-dev html-webpack-plugin
```

配置：

```
title: 用来生成页面的 title 元素
filename: 输出的 HTML 文件名，默认是 index.html, 也可以直接配置带有子目录。
template: 模板文件路径，支持加载器，比如 html!./index.html
inject: true | 'head' | 'body' | false  ,注入所有的资源到特定的 template 或者
templateContent 中，如果设置为 true 或者 body，所有的 javascript 资源将被放置到 body
元素的底部，'head' 将放置到 head 元素中。
favicon: 添加特定的 favicon 路径到输出的 HTML 文件中。
minify: {} | false , 传递 html-minifier 选项给 minify 输出
hash: true | false, 如果为 true, 将添加一个唯一的 webpack 编译 hash 到所有包含的脚本和
CSS 文件，对于解除 cache 很有用。
cache: true | false，如果为 true, 这是默认值，仅仅在文件修改之后才会发布文件。
showErrors: true | false, 如果为 true, 这是默认值，错误信息会写入到 HTML 页面中
chunks: 允许只添加某些块 (比如，仅仅 unit test 块)
chunksSortMode: 允许控制块在添加到页面之前的排序方式，支持的值：'none' | 'default' |
{function}-default:'auto'
excludeChunks: 允许跳过某些块，(比如，跳过单元测试的块)
```

案例：

```
const path = require("path");
const htmlWebpackPlugin = require("html-webpack-plugin");

module.exports = {
 ...
  plugins: [
    new htmlWebpackPlugin({
      title: "My App",
      filename: "app.html",
      template: "./src/index.html"
    })
  ]
};


//index.html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title><%= htmlWebpackPlugin.options.title %></title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

clean-webpack-plugin

```
npm install --save-dev clean-webpack-plugin
```

```
const { CleanWebpackPlugin } = require("clean-webpack-plugin");

...

plugins: [
    new CleanWebpackPlugin()
]
```

clean-webpack-plugin:如何做到dist目录下某个文件或目录不被清空： 使用配置项:cleanOnceBeforeBuildPatterns 案例：cleanOnceBeforeBuildPatterns: ["**/*", "!dll", "!dll/**"], ！ 感叹号相当于exclude 排除，意思是清空操作排除dll目录，和dll目录下所有文件。 注意：数组列表里的“*/”是默认值，不可忽略，否则不做清空操作。