###前端模块化介绍

\* 为什么需要模块化

前端页面模块化主要是通过引入一大堆script标签,他有以下缺点:

> \* 网页体积变得臃肿

> \* 产生大量的HTTP请求

> \* 难以维护

而打包工具可以解决这些问题,他可以自动加载各个模块,管理脚本之间的依赖关系,webpack就是这样一款工具,是一个现代JavaScript应用程序的静态模块打包器

### webpack概念

https://webpack.docschina.org/concepts

### webpack安装和测试

\* 安装

```

mkdir project && cd project

npm init --yes

npm install webpack webpack-cli --save-dev

```

> \* 注意点

>> \* --save-dev 为本地安装,推荐使用,这样容易分别升级项目,依赖会加在package.json的devDependencies中,一般是辅助开发的依赖,不会打包上线的

>> \* --save 依赖会加在package.json的dependencies中,一般是需要最终需要打包到业务代码中的依赖

>> \* --global 为全局安装

>> \* 也可以用npm install --save-dev webpack@<version>来指定安装的版本号

>> \* npx webpack -v 查看版本号

\* 构建项目文件并测试

> \* 新增以下目录和文件

```

project

-dist

index.html(该文件使用main.js文件)

-src

index.js(测试的js文件)

```

> \* 打包

```

npx webpack

```

> \* 注意点

>> \* npx 命令,可以运行在./node\_modules/.bin/中文件

>> \* npx 运行后会在dist目录下生成打包后的文件main.js

>> \* webpack4.0以上不用配置文件也可以打包,但不推荐使用

\* 在项目目录下配置文件 webpack.config.js

```

const path = require('path');

module.exports = {

mode:'development',

entry: './src/index.js',

output: {

filename: 'bundle.js',

path: path.resolve(\_\_dirname, 'dist')

}

};

```

> \* 新增配置文件后打包

```

npx webpack --config webpack.config.js

```

> \* 注意点

>> \* 虽然没有配置文件webpack也可以打包,但通常都需要配置文件来处理复杂项目

>> \* webpack默认的配置文件是webpack.config.js,也可以指定为其他的,如果使用默认的话,--config webpack.config.js可以省略

>> \* npx 运行后会在dist目录下生成打包后的文件bundle.js

### webpack指南

https://webpack.docschina.org/guides

### loader

webpack自身只支持JavaScript

而loader能够让webpack处理那些非JavaScript文件

### 加载css

\* 安装相关的loader

```

npm install --save-dev style-loader css-loader

```

\* 修改配置文件

```javascript

module: {

rules: [

{

test: /\.css$/,

use: [

'style-loader',

'css-loader'

]

}

]

}

```

### 加载图片

\* 安装相关的loader

```

npm install --save-dev url-loader

```

\* 修改配置文件

```javascript

module:{

rules:[

{

test:/\.css$/,

use: [

'style-loader',

'css-loader'

]

},

{

test:/\.(png|jpg|gif)$/,

use:[

'url-loader'

]

}

]

},

```

\* 自动打包,有更新就会自动执行打包

```

npx webpack --watch

```

> 注意,如果修改了配置文件需要从新启动

\* 显示打包过程

```

npx webpack --progress

```

\* 显示打包模块

```

npx webpack --display-modules

```

\* 显示打包原因

```

npx webpack --display-reasons

```

\* 添加npm脚本(package.json)

```

"scripts": {

"build": "webpack --progress --display-modules --display-reasons --watch"

},

```

添加脚本后用下面命令打包

> npm run build

### 入口

\* 入口起点(entry point)指示 webpack 应该使用哪个模块，来作为构建其内部依赖图的开始

\* 入口分为单入口(一个入口起点)和多入口(多个入口起点)

\* 单入口单页应用(SPA),多入口多页应用(MPA)

\* 单入口写法

```

entry: {main:'./src/index.js'},

```

可以简写为

```

entry: './src/index.js',

```

\* 多入口写法

```

entry: {

index: "./index.js",

about: "./about.js",

contact: "./contact.js"

}

```

### 出口

\* 出口用来控制打包编译后的文件

\* 对于单个入口,出口可以是一个静态文件

\* 如果有多个入口起点,也只指定一个输出配置,应该用模板来描述出口文件的名称

\* 常见的模板有

> \* [hash] 模块标识符的hash

> \* [chunkhash] chunk内容的hash

> \* [name] chunk名称

```

output: {

filename: '[name]-[hash].bundle.js',

path: path.resolve(\_\_dirname, 'dist')

},

```

\* 注意点:

> \* fiflenam不用带路径

> \* path用绝对路径

### 自动生成HTML

\* 安装插件

```

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

```

\* 修改配置文件使用插件

```

const htmlWebpackPlugin = require('html-webpack-plugin')

plugins:[

new htmlWebpackPlugin({

template:'./src/view/index.html',//模板文件

inject:'head'//脚本写在那个标签里,默认是true(在body结束后)

hash:true//给生成的js/css文件添加一个唯一的hash

})

]

```

### 清理文件

\* 安装插件

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

\* 修改配置文件

plugins:[

new htmlWebpackPlugin({

template:'./src/view/index.html',//模板文件

filename:'main.html',//生成的文件名

inject:'head',//脚本写在那个标签里,默认是true(在body结束后)

hash:true//

}),

new CleanWebpackPlugin(['dist'])

]

### webpack-dev-server

> webpack-dev-server提供了一个简单的基于node express的web服务器,能够实时重新加载页面

\* 安装

npm install --save-dev webpack-dev-server

\* 修改配置文件

devServer:{

contentBase: './dist'

}

\* 启动

npx webpack-dev-server

或者修改package.json

"scripts": {

"build": "webpack --progress --display-modules --display-reasons --watch",

"start": "webpack-dev-server --open"

},

后使用

npm start 启动