Webpack-Day1

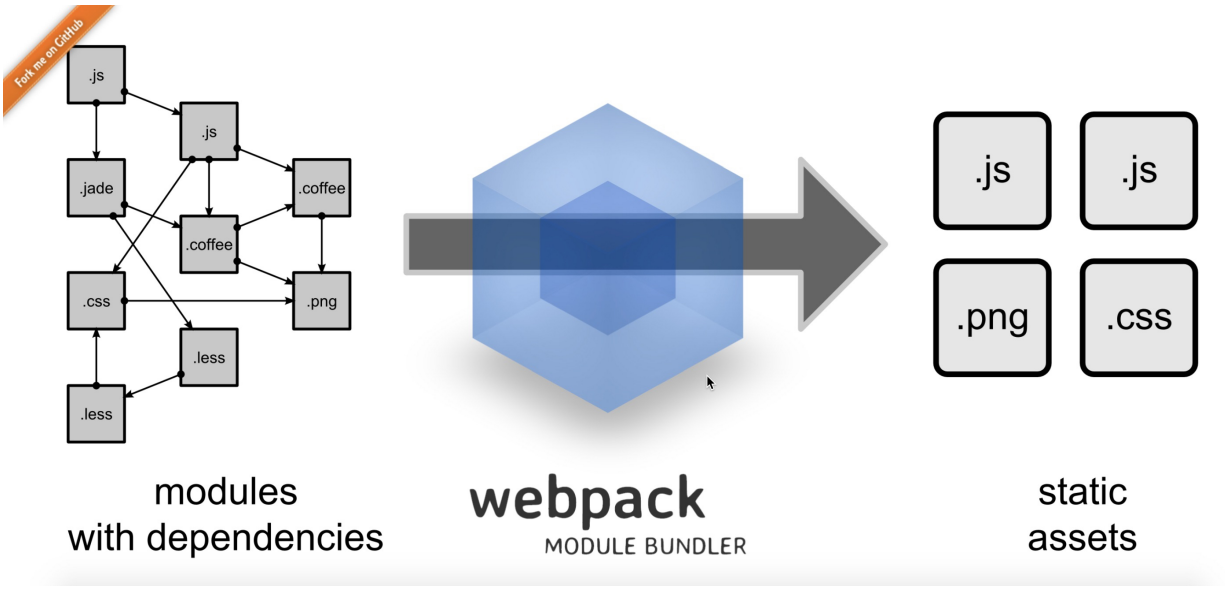
知识要点

- 清楚理解和掌握webpack的配置、概念

文档

- 官方网站：https://webpack.js.org/

1.webpack简介



webpack is a module bundler(模块打包工具)

`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.安装webpack

2.1-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

2.2-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

2.3-项目安装 推荐

```
# 安装最新的稳定版本
npm i -D webpack

# 安装指定版本
npm i -D webpack@<version>

# 安装最新的体验版本 可能包含bug,不要用于生产环境
npm i -D webpack@beta

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

2.4-检查安装

```
webpack -v //command not found 默认在全局环境中查找

npx webpack -v// npx帮助我们在项目中的node_modules里查找webpack
./node_modules/.bin/webpack -v//到当前的node_modules模块里指定webpack
```

3.启动webpack执行构建

1Chunk = 1bundle

chunk是代码块

bundle是资源文件

1个chunk可以是多个模块组成的

模块？nodeJS里 万物皆模块

数组：

webpack会自动生成另外一个入口模块，并将数组中的每个指定的模块加载进来，并将最后一个模块的module.exports作为入口模块的module.exports导出。

index-782c94

other-425b0f

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

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

选项	描述
`development`	会将 `DefinePlugin` 中 `process.env.NODE_ENV` 的值设置为 `development` 。启用 `NamedChunksPlugin` 和 `NamedModulesPlugin` 。
`production`	会将 `DefinePlugin` 中 `process.env.NODE_ENV` 的值设置为 `production` 。启用 `FlagDependencyUsagePlugin` , `FlagIncludedChunksPlugin` , `ModuleConcatenationPlugin` , `NoEmitOnErrorsPlugin` , `OccurrenceOrderPlugin` , `SideEffectsFlagPlugin` 和 `TerserPlugin` 。
`none`	退出任何默认优化选项

如果没有设置，webpack 会将 `mode` 的默认值设置为 `production` 。模式支持的值为：

> *记住，设置 `NODE_ENV` 并不会自动地设置 `mode` 。*

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

Webpack 默认只支持.json 和 .js模块 不支持 不认识其他格式的模块

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
    }
  }
  ]
}
```

- loader: file-loader：处理静态资源模块

  loader: file-loader

  原理是把打包入口中识别出的资源模块，移动到输出目录，并且返回一个地址名称

  所以我们什么时候用file-loader呢?

  场景：就是当我们需要模块，仅仅是从源代码挪移到打包目录，就可以使用file-loader来处理，txt，svg，csv，excel，图片资源啦等等

  ```
  npm install file-loader -D
  ```

  案例：

```
module: {
    rules: [
      {
        test: /\.(png|jpe?g|gif)$/,
        //use使用一个loader可以用对象，字符串，两个loader
需要用数组
        use: {
          loader: "file-loader",
          //   options额外的配置，比如资源名称
          options: {
            //  placeholder 占位符  [name]老资源模块的
名称
            //   [ext]老资源模块的后缀
            // https://webpack.js.org/loaders/file-
loader#placeholders
            name: "[name]_[hash].[ext]",
            //打包后的存放位置
            outputPath: "images/"
          }
        }
      }
    ]
  },
```

```
import pic from "./logo.png";

var img = new Image();
img.src = pic;
img.classList.add("logo");

var root = document.getElementById("root");
root.append(img);
```

- 处理字体 https://www.iconfont.cn/?spm=a313x.7781069.1998910419.d4d0a486a

```
//css
@font-face {
  font-family: "webfont";
  font-display: swap;
  src: url("webfont.woff2") format("woff2");
}

body {
  background: blue;
  font-family: "webfont" !important;
}


//webpack.config.js
{
  test: /\.(eot|ttf|woff|woff2|svg)$/,
  use: "file-loader"
}
```

- url-loader file-loader加强版本

  url-loader内部使用了file-loader,所以可以处理file-loader所有的事情，但是遇到jpg格式的模块，会把该图片转换成base64格式字符串，并打包到js里。对小体积的图片比较合适，大图片不合适。

```
npm install url-loader -D
```

案例；

```
module: {
    rules: [
      {
        test: /\.(png|jpe?g|gif)$/,
        use: {
          loader: "url-loader",
          options: {
            name: "[name]_[hash].[ext]",
            outputPath: "images/",
            //小于2048，才转换成base64
            limit: 2048
          }
        }
      }
    ]
  },
```

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"]
}
```

```
{
  test:/\.css$/,
    use:[{
      loader:"style-loader",
      options: {
              injectType: "singletonStyleTag" // 将所有的style标签合并成一个
            }
    },"css-loader"]
}
```

Less样式处理

less-load 把less语法转换成css

```
$ npm install less less-loader --save-dev
```

案例：

loader有顺序，从右到左，从下到上

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

样式自动添加前缀：

https://caniuse.com/

Postcss-loader

```
npm i  postcss-loader autoprefixer -D
```

新建postcss.config.js

```
//webpack.config.js
{
  test: /\.css$/,
```

```
  use: ["style-loader", "css-loader", "postcss-
loader"]
},

//postcss.config.js
module.exports = {
  plugins: [
    require("autoprefixer")({
      overrideBrowserslist: ["last 2 versions",
">1%"]
    })
  ]
};
```

loader 处理webpack不支持的格式文件，模块

一个loader只处理一件事情

loader有执行顺序

5.Plugins

- 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()
]
```