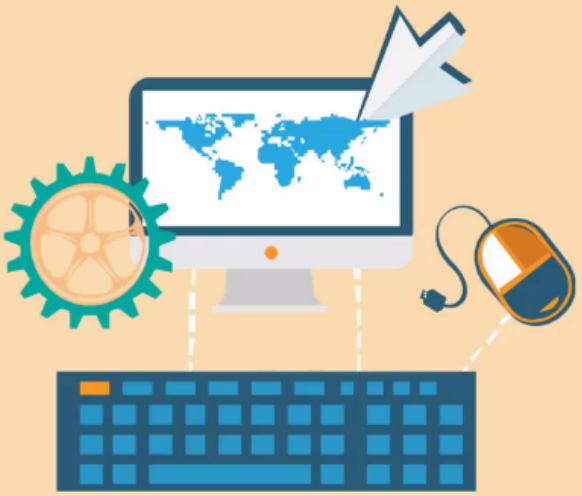
库打包工具rollup使用

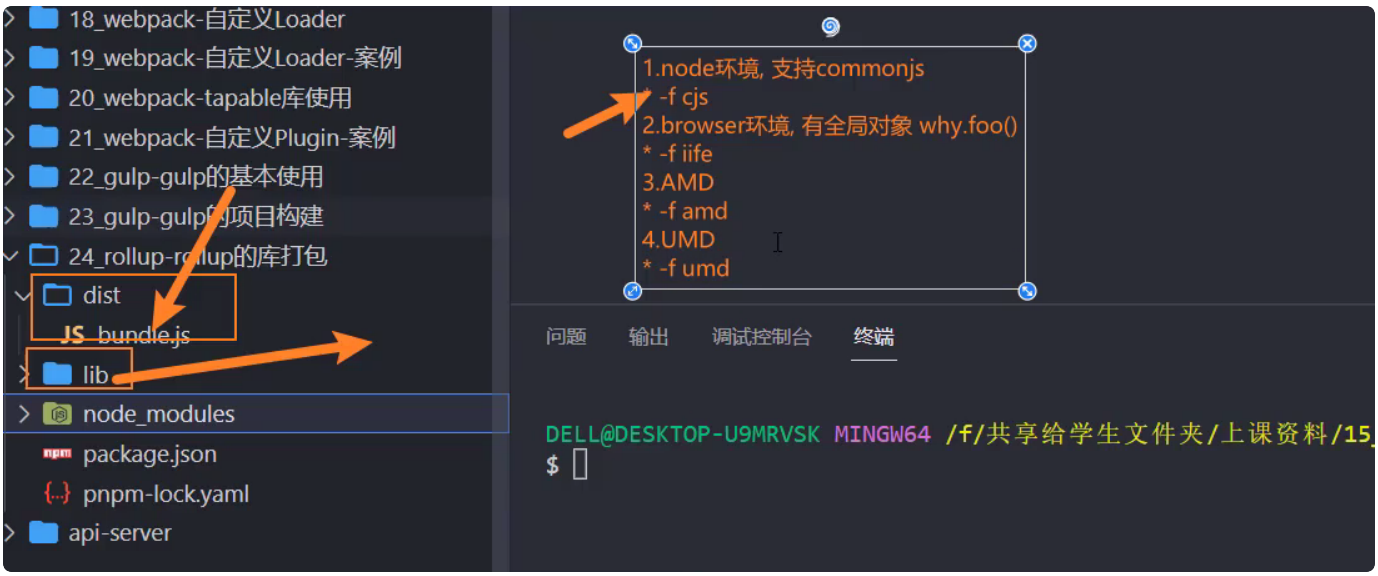
王红元 coderwhy

目录
content



认识rollup

- **我们来看一下官方对rollup的定义：**
  - Rollup is a module bundler for JavaScript which compiles small pieces of code into something larger and more complex, such as a library or application.
  - Rollup是一个JavaScript的模块化打包工具，可以帮助我们编译小的代码到一个大的、复杂的代码中，比如一个库或者一个应用程序
- **我们会发现Rollup的定义、定位和webpack非常的相似：**
  - Rollup也是一个模块化的打包工具，但是Rollup主要是针对ES Module进行打包的；
  - 另外webpack通常可以通过各种loader处理各种各样的文件，以及处理它们的依赖关系；
  - rollup更多时候是专注于处理JavaScript代码的（当然也可以处理css、font、vue等文件）；
  - 另外rollup的配置和理念相对于webpack来说，更加的简洁和容易理解；
  - 在早期webpack不支持tree shaking时，rollup具备更强的优势；
- **目前webpack和rollup分别应用在什么场景呢?**
  - 通常在实际项目开发过程中，我们都会使用webpack（比如react、angular项目都是基于webpack的）；
  - 在对库文件进行打包时，我们通常会使用rollup（比如vue、react、dayjs源码本身都是基于rollup的，Vite底层使用Rollup）；



Rollup基本使用

- 我们可以先安装rollup：

```
# 全局安装
npm install rollup -g
# 局部安装
npm install rollup -D
```

- 创建main.js文件，打包到bundle.js文件中：

```
# 打包浏览器的库
npx rollup ./src/main.js -f iife -o dist/bundle.js

# 打包AMD的库
npx rollup ./src/main.js -f amd -o dist/bundle.js

# 打包CommonJS的库
npx rollup ./src/main.js -f cjs -o dist/bundle.js

# 打包通用的库（必须跟上name）
npx rollup ./src/main.js -f umd --name mathUtil  -o dist/bundle.js
```

Rollup的配置文件

- 我们可以将配置信息写到配置文件中rollup.config.js文件：

```
module.exports = {
  input: "./src/index.js",
  output: {
    format: "umd",
    name: "whyUtils",
    file: "dist/why.umd.js"
  }
}
```

- 我们可以对文件进行分别打包，打包出更多的库文件（用户可以根据不同的需求来引入）：

```
export default {
  input: "./src/main.js",
  output: [
    {
      format: "cjs",
      file: "dist/why.common.js",
    },
    {
      format: "amd",
      file: "dist/why.amd.js"
    },
```

解决commonjs和第三方库问题

- 安装解决commonjs的库：

```
npm install @rollup/plugin-commonjs -D
```

- 安装解决node_modules的库：

```
npm install @rollup/plugin-node-resolve -D
```

- 打包和排除lodash

```
output: {
  format: "umd",
  file: "dist/why.umd.js",
  name: "whyUtil",
  globals: {
    "lodash": "_"
  },
},
external: ["lodash"],
```

Babel转换代码

- 如果我们希望将ES6转成ES5的代码，可以在rollup中使用babel。
- 安装rollup对应的babel插件：

```
npm install @rollup/plugin-babel -D
```

- 修改配置文件：
  - 需要配置babel.config.js文件；
  - babelHelpers:

```
resolve(),
babel({
  exclude: "node_modules/**",
  babelHelpers: "bundled",
}),
commonjs(),
```

```
module.exports = {
  presets: [
    "@babel/preset-env",
    "@babel/preset-react"
  ]
}
```

Teser代码压缩

- 如果我们希望对代码进行压缩，可以使用@rollup/plugin-terser：

```
npm install @rollup/plugin-terser -D
```

- 配置terser：

```
const plugins = [
  resolve(),
  babel({
    exclude: "node_modules/**",
    babelHelpers: "bundled"
  }),
  commonjs(),
  terser(),
]
```

处理css文件

- 如果我们项目中需要处理css文件，可以使用postcss：

```
npm install rollup-plugin-postcss postcss -D
```

- 配置postcss的插件：

```
const plugins = [
  resolve(),
  babel({
    exclude: "node_modules/**",
    babelHelpers: "bundled"
  }),
  commonjs(),
  postcss(),
  terser(),
]
```

处理vue文件

- 处理vue文件我们需要使用rollup-plugin-vue插件：
  - 但是注意：默认情况下我们安装的是vue3.x的版本，所以我这里指定了一下rollup-plugin-vue的版本；

```
npm install rollup-plugin-vue @vue/compiler-sfc -D
```

- 使用vue的插件：

```
resolve(),
babel({
  exclude: "node_modules/**",
  babelHelpers: "bundled",
}),
commonjs(),
postcss(),
vue()
```

打包vue报错

- 在我们打包vue项目后，运行会报如下的错误：

```
Uncaught ReferenceError: process is not defined
    at why.umd.js:6
    at why.umd.js:1
    at why.umd.js:1
(anonymous) @ why.umd.js:6
(anonymous) @ why.umd.js:1
(anonymous) @ why.umd.js:1
```

- 这是因为在我们打包的vue代码中，用到 process.env.NODE_ENV，所以我们可以使用一个插件 rollup-plugin-replace 设置它对应的值：

```
npm install @rollup/plugin-replace -D
```

- 配置插件信息：

```
resolve(),
commonjs(),
replace({
  'process.env.NODE_ENV': JSON.stringify( 'production' )
}),
vue(),
```

搭建本地服务器

- 第一步：使用rollup-plugin-serve搭建服务

```
npm install rollup-plugin-serve -D
```

```
serve({
  // open: true,
  port: 8888,
  contentBase: "."
}),
```

- 第二步：当文件发生变化时，自动刷新浏览器

```
npm install rollup-plugin-livereload -D
```

```
livereload()
```

- 第三步：启动时，开启文件监听

```
npx rollup -c -w
```

区分开发环境

- 我们可以在package.json中创建一个开发和构建的脚本：

```
"scripts": {
  "serve": "rollup -c --environment NODE_ENV:development -w",
  "build": "rollup -c --environment NODE_ENV:production"
},
```

```
const isProduction = process.env.NODE_ENV === "production";

const plugins = [···
]

if (isProduction) {···
} else {···
}
```