构建工具@李立超

构建工具

构建工具可以对代码进行打包，将多个模块整合成一个文件，并将使用 ESM 规范编写的代码转换为旧版的 JavaScript 语法。这样既解决了浏览器对模块化规范的不兼容问题，又解决了模块过多导致的性能问题（浏览器需要分别请求所有模块，浪费时间）。常用的构建工具有 Webpack 和 Vite。

Webpack

快速使用

1. 初始化项目：`yarn init -y`
2. 安装依赖：`yarn add -D webpack webpack-cli`

   > `webpack` 是实际执行打包任务的工具，而 `webpack-cli` 是一个辅助工具，提供了命令行界面，让使用 Webpack 更加方便。此外，注意这里安装的是开发依赖。
3. 在项目中创建 `src` 目录，并在其中创建 `index.js` 文件
4. 执行以下命令对项目进行打包：`yarn webpack`

   > 此时，`src/index.js` 及其所需的所有模块都会被打包到 `dist/main.js`。

配置文件 webpack.config.js

项目中的 `webpack.config.js` 是 webpack 的配置文件。一般的，`src` 文件夹下存放项目的原始代码，`dist` 文件夹下存放项目打包后的代码。

> 注意 `webpack.config.js` 需要遵循 CommonJS 模块化规范，即使用 `require` 导入模块。`src` 中的代码是前端代码，需要遵循 ESM 模块化规范，即使用 `import` 导入模块。

mode

`mode` 配置项用于设置 webpack 的打包模式。

```
module.exports = {
    mode: "production" // "production" 表示生产模式，"development" 表示开发模式
}
```

entry

`entry` 配置项用于指定 webpack 打包时的主文件。默认值为 `./src/index.js`。该配置项有三种配置方式，字符串、数组或对象。

```
module.exports = {
    // 1. 【string】指定单个文件进行打包 => 一个文件
    entry: "./hello/hello.js", // 文件打包到 ./dist/main.js
    // 2. 【array】指定多个文件进行打包 => 一个文件
    entry: ["./src/a.js", "./src/b.js"], // 文件打包到 ./dist/main.js
    // 3. 【object】指定多个文件分别打包 => 多个文件
    entry: { // key 指定打包后的文件名，value 指定打包文件
        dist_a: "./src/a.js", // 文件打包到 ./dist/dist_a.js
        dist_b: "./src/b.js" // 文件打包到 ./dist/dist_b.js
    }
}
```

output

`output` 配置项用于指定 webpack 代码打包后的地址。 默认值为 `./dist/main.js`。该配置项可以由 `path`、`filename`、`clean` 等子配置组成。

```
const path = require("path");

module.exports = {
    output: {
        // output.path 指定打包的目录（必须是绝对路径）
        path: path.resolve(__dirname, "dist"),
        // output.path 指定打包后的文件名
        // 1. 指定单个打包文件名
        filename: "main.js",
        // 2. 指定多个打包文件名，其中 [xxx] 由 webpack 自动指定。
        filename: "[name]-[id]-[hash].js",
        // output.path 指定打包前是否自动清理打包目录中的内容
        clean: true
    }
}
```

loader

webpack 默认只会打包 `js` 文件，如果想让其打包其他类型的文件，则要配置对应的 `loader`。该效果通过 `module` 配置项实现，其中指定了一组 `rules` 用于匹配文件、指定要使用的 `loader`。

```
module.exports = {
    module: [
        /*  使用 css-loader 和 style-loader 打包 .css 文件
            - 需要预先安装对应的 loader，如
                yarn add css-loader style-loader -D
            - 通过 use 配置 loader 需要注意顺序，数组中的 loader 从后往前依次处理
            - css-loader 用于打包 .css 文件，style-loader 用于应用 .css 文件
            - 在 src/index.js 中可以通过 import ".css 文件路径" 的方式引入 .css 文件
        */
        {test: /\.css$/i, use: ["style-loader", "css-loader"]},
        /*  webpack 认为图片是直接资源类型的数据，因此不用额外下载 loader。
            - 对于直接资源类型，可以通过 type 指定资源类型，webpack 就会使用其内置的 loader 进行打包。
            - 在 src/index.js 中可以通过 import An from "图片路径" 的方式引入图片文件
        */
        {test: /\.jpg$/i, type: "asset/resource"}
    ]
}
```

babel

1. 解释：babel 是一个工具，可以将新的 js 语法转换为旧的 js 语法，从而提高代码的兼容性。
2. 使用：如果希望在 webpack 中支持 `babel`，则需要向 webpack 中引入对应的 `loader`。
    a. 安装 `yarn add -D babel-loader @babel/core @babel/preset-env`
    b. 配置

```
module.exports = {
    module: {
        rules: [
            {
                test: /\.m?js$/,
                exclude: /(node_modules|bower_components)/,
                use: {
                    loader: 'babel-loader',
                    options: {
                        presets: ['@babel/preset-env'],
                    },
                },
            },
        ];
    }
}
```

3. babel 的兼容性配置：可以在 `package.json` 中的 `browserslist` 配置项中配置 babel 转换后的 js 代码要兼容的浏览器。

plugins

`plugins` 配置项为 webpack 添加插件，实现其功能的扩展。

html-webpack-plugin

1. 解释：`html-webpack-plugin` 插件用于在 webpack 完成打包后，自动在打包目录 `./dist/` 生成 html 文件。
2. 使用
   a. 安装 `yarn add -D html-webpack-plugin`
   b. 配置

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

module.exports = {
    plugins: [
        /*  通过 new 创建一个插件实例，结合 plugins 配置项即可使用插件
            - title 用于指定要生成的 HTML 文件的 <title> 标签的值
            - template 用于指定要生成的 HTML 文件的模板，即根据已有的 HTML 文件，生成一个相同的文件
        */
        new HtmlPlugin({
            title: "Hello Webpack",
            template: "html 文件模板地址"
        })
    ]
}
```

代码运行

1. 本地运行：`yarn webpack --watch`。此时一旦源代码 `./src/` 更新后，webpack 就会自动打包并将其保存到 `./dist/`。
2. 服务器运行：`yarn add -D webpack-dev-server`
   a. `yarn webpack serve` 将打包后的代码部署到服务器。一旦源代码 `./src/` 更新，webpack 就会重新打包并将其部署到服务器。即本地代码的改变，会对应服务器页面实时刷新。
   b. `yarn webpack serve --open` 将打包后的代码部署到服务器，并自动在浏览器中打开。同时，本地代码的改变，也会对应服务器页面实时刷新。

c. 注意：由于打包后的代码部署在服务器中，因此最后仍然需要使用 `yarn webpack` 将代码打包并保存到 `./dist/`。

代码调试 —— sourceMap

在 `webpack.config.js` 中可以进行如下配置，设置代码的映射方式。此时，在网页中 debug 时，可以对未打包的源代码进行调试，而运行在网页中的代码是打包后的代码（可读性差）。

```
module.exports = {
    devtool: "inline-source-map"
}
```

Vite

快速使用

1. 初始化项目：`yarn init -y`
2. 安装依赖：`yarn add -D vite`
3. 相关命令
   a. `yarn vite` 启动一个开发服务器，并在服务器中运行源代码（没有打包）
   b. `yarn vite build` 将源代码打包并保存到项目根目录的 `dist` 文件夹中

      > 打包后的项目需要部署在服务器的根目录才能运行

   c. `yarn vite preview` 启动一个服务器，预览 `dist` 文件夹中打包后的代码

> 可以在 `package.json` 中通过 `scripts` 配置上述命令如下，从而简化 vite 命令
>
> ```
> "scripts": {
>     "dev": "vite", // yarn dev
>     "build": "vite build", // yarn build
>     "preview": "vite preview" // yarn preview
> }
> ```

Vite 与 Webpack 的对比

- Vite 基于 ES6 模块化规范，因此其配置文件 `vite.config.js` 中应该使用 `export default xxx` 形式的语法。同时，源代码中必须使用 `<script type="module" src="xxx"></script>` 的方式引入 js 代码。
- Vite 支持在项目根目录下编写源代码，打包后的内容存放在根目录的 `dist` 文件夹下。

- Vite 在开发时不打包代码，而是采用 ESM 的方式来运行项目。在部署时才对项目进行打包。
- Vite 更快、使用更加方便。

快速搭建一个 Vite 项目

```
1 yarn create vite
```

配置文件 vite.config.js

plugins

`plugins` 配置项为 vite 添加插件，实现其功能的扩展。

@vitejs/plugin-legacy

1. 解释：该插件类似 webpack 中的 `babel-loader` ，用于为传统浏览器提供支持。
2. 使用
   a. 安装 `yarn add -D @vitejs/plugin-legacy terser`
   b. 配置

```
1  // vite.config.js
2  // 引入插件
3  import legacy from '@vitejs/plugin-legacy'
4  // 也可以直接暴露出一个配置对象，使用 defineConfig 函数的好处是配置有提示
5  import { defineConfig } from 'vite'
6
7  export default defineConfig({
8    plugins: [
9      legacy({ // 使用插件
10       targets: ['defaults', 'not IE 11'],
11     }),
12   ],
13 })
```