devServer和VueCLI

王红元 coderwhy

为什么要搭建本地服务器？

- **目前我们开发的代码，为了运行需要有两个操作：**
  - 操作一：npm run build，编译相关的代码；
  - 操作二：通过live server或者直接通过浏览器，打开index.html代码，查看效果；
- 这个过程经常操作会影响我们的开发效率，我们希望可以做到，当**文件发生变化**时，可以**自动的完成 编译 和 展示**；
- **为了完成自动编译，webpack提供了几种可选的方式：**
  - webpack watch mode；
  - webpack-dev-server（常用）；
  - webpack-dev-middleware；

Webpack watch

- webpack给我们提供了**watch模式**：
  - 在该模式下，webpack依赖图中的所有文件，只要有一个发生了更新，那么代码将被重新编译；
  - 我们不需要手动去运行 npm run build指令了；
- **如何开启watch呢？两种方式：**
  - 方式一：在导出的配置中，添加 watch: true；
  - 方式二：在启动webpack的命令中，添加 --watch的标识；
- **这里我们选择方式二**，在package.json的 scripts 中添加一个 watch 的脚本：

```
"scripts": {
  "build": "webpack --config wk.config.js",
  "watch": "webpack --watch",
  "type-check": "tsc --noEmit",
  "type-check-watch": "npm run type-check -- --watch"
},
```

webpack-dev-server

- 上面的方式**可以监听到文件的变化**，但是事实上它本身是**没有自动刷新浏览器的功能**的：
  - 当然，目前我们可以在VSCode中使用live-server来完成这样的功能；
  - 但是，我们希望在不使用live-server的情况下，可以具备live reloading（实时重新加载）的功能；
- **安装webpack-dev-server**

```
npm install webpack-dev-server -D
```

- **修改配置文件**，告知 dev server，从什么位置查找文件：

```
devServer: {
  contentBase: "./build"
},
```

```
target: "web",
```

```
"serve": "webpack serve --config wk.config.js",
```

- **webpack-dev-server 在编译之后不会写入到任何输出文件，而是将 bundle 文件保留在内存中：**
  - 事实上webpack-dev-server使用了一个库叫memfs（memory-fs webpack自己写的）

认识模块热替换（HMR）

- **什么是HMR呢？**
  - HMR的全称是Hot Module Replacement，翻译为模块热替换；
  - 模块热替换是指在 应用程序运行过程中，替换、添加、删除模块，而无需重新刷新整个页面；
- **HMR通过如下几种方式，来提高开发的速度：**
  - 不重新加载整个页面，这样可以保留某些应用程序的状态不丢失；
  - 只更新需要变化的内容，节省开发的时间；
  - 修改了css、js源代码，会立即在浏览器更新，相当于直接在浏览器的devtools中直接修改样式；
- **如何使用HMR呢？**
  - 默认情况下，webpack-dev-server已经支持HMR，我们只需要开启即可；
  - 在不开启HMR的情况下，当我们修改了源代码之后，整个页面会自动刷新，使用的是live reloading；

开启HMR

- 修改webpack的配置：

```
devServer: {
  hot: true
},
```

- 浏览器可以看到如下效果：

```
[HMR] Waiting for update signal from WDS...
[WDS] Hot Module Replacement enabled.
[WDS] Live Reloading enabled.
```

- 但是你会发现，**当我们修改了某一个模块的代码时，依然是刷新的整个页面：**
  - 这是因为我们需要去指定哪些模块发生更新时，进行HMR；

```
if (module.hot) {
  module.hot.accept("./util.js", () => {
    console.log("util更新了");
  })
}
```

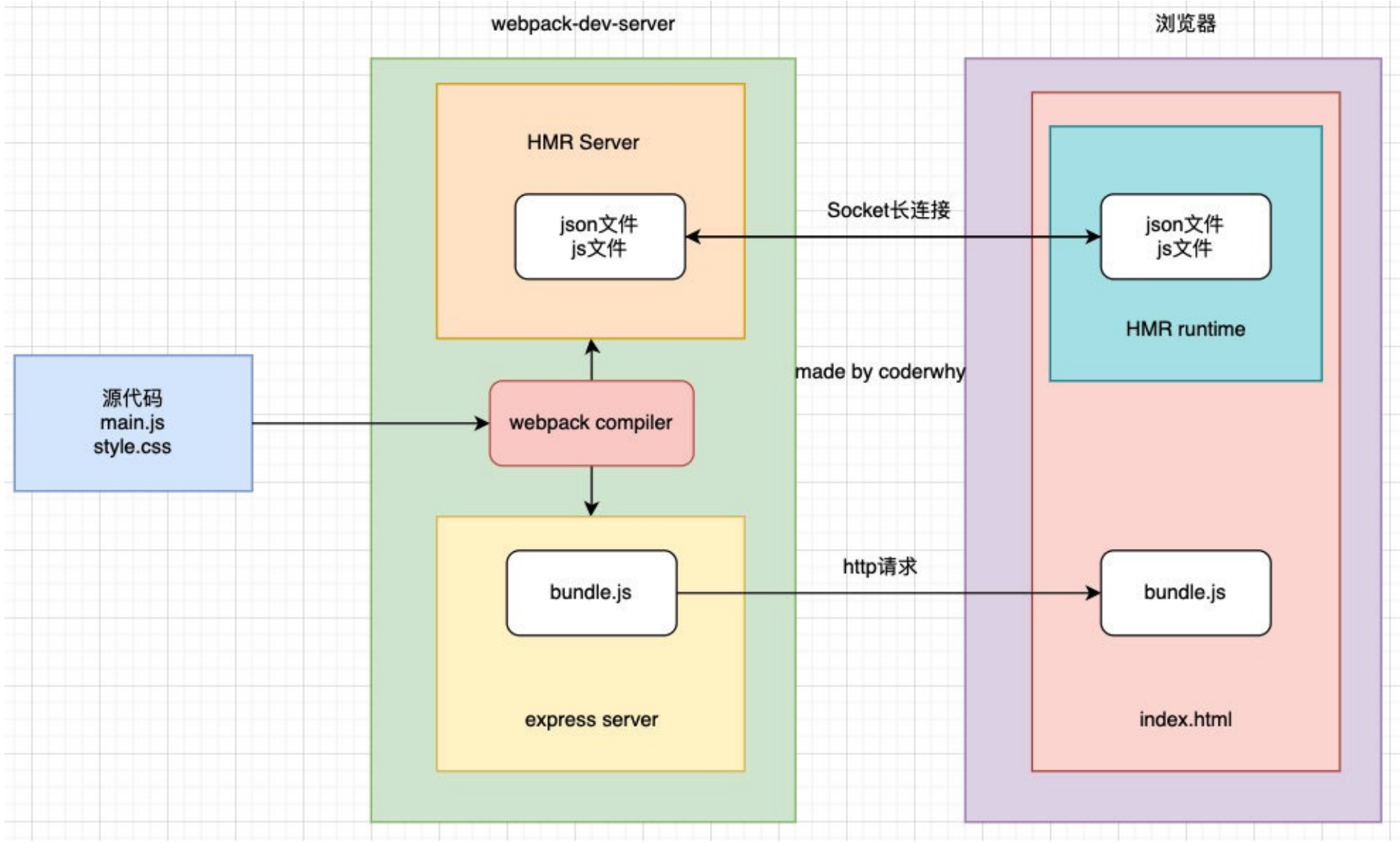
框架的HMR

- **有一个问题：**在开发其他项目时，**我们是否需要经常手动去写入 module.hot.accpet相关的API呢**？
  - 比如开发Vue、React项目，我们修改了组件，希望进行热更新，这个时候应该如何去操作呢？
  - 事实上社区已经针对这些有很成熟的解决方案了；
  - 比如vue开发中，我们使用vue-loader，此loader支持vue组件的HMR，提供开箱即用的体验；
  - 比如react开发中，有React Hot Loader，实时调整react组件（目前React官方已经弃用了，改成使用react-refresh）；
- 接下来我们来演示一下Vue实现一下HMR功能。

HMR的原理

- **那么HMR的原理是什么呢？如何可以做到只更新一个模块中的内容呢？**
  - webpack-dev-server会创建两个服务：提供静态资源的服务（express）和Socket服务（net.Socket）；
  - express server负责直接提供静态资源的服务（打包后的资源直接被浏览器请求和解析）；
- **HMR Socket Server，是一个socket的长连接：**
  - 长连接有一个最好的好处是建立连接后双方可以通信（服务器可以直接发送文件到客户端）；
  - 当服务器监听到对应的模块发生变化时，会生成两个文件.json（manifest文件）和.js文件（update chunk）；
  - 通过长连接，可以直接将这两个文件主动发送给客户端（浏览器）；
  - 浏览器拿到两个新的文件后，通过HMR runtime机制，加载这两个文件，并且针对修改的模块进行更新；

HMR的原理图

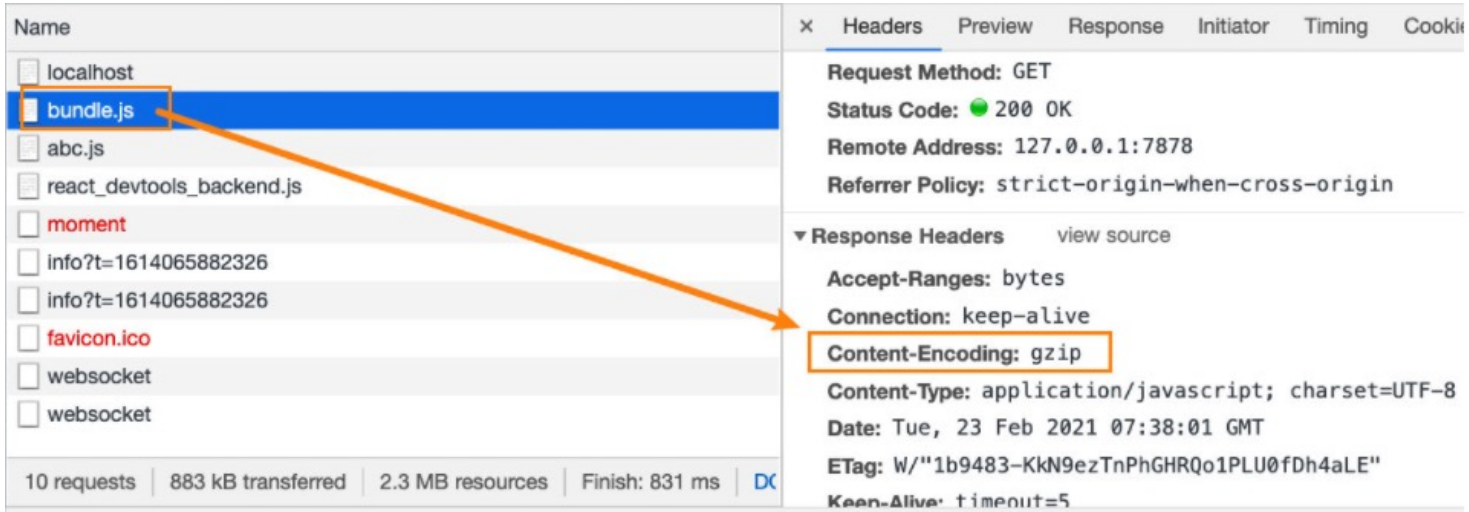

webpack-dev-server
浏览器
HMR Server
json文件
js文件
Socket长连接
json文件
js文件
HMR runtime
源代码
main.js
style.css
webpack compiler
made by coderwhy
bundle.js
http请求
bundle.js
express server
index.html

hotOnly、host配置

- **host设置主机地址：**
  - 默认值是localhost；
  - 如果希望其他地方也可以访问，可以设置为 0.0.0.0；
- **localhost 和 0.0.0.0 的区别：**
  - localhost：本质上是一个域名，通常情况下会被解析成127.0.0.1;
  - 127.0.0.1：回环地址(Loop Back Address)，表达的意思其实是我们主机自己发出去的包，直接被自己接收;
    - ✓ 正常的数据库包经常 应用层 - 传输层 - 网络层 - 数据链路层 - 物理层；
    - ✓ 而回环地址，是在网络层直接就被获取到了，是不会经常数据链路层和物理层的;
    - ✓ 比如我们监听 127.0.0.1时，在同一个网段下的主机中，通过ip地址是不能访问的;
  - 0.0.0.0：监听IPV4上所有的地址，再根据端口找到不同的应用程序;
    - ✓ 比如我们监听 0.0.0.0时，在同一个网段下的主机中，通过ip地址是可以访问的;

port、open、compress

- **port设置监听的端口，默认情况下是8080**
- **open是否打开浏览器：**
  - 默认值是false，设置为true会打开浏览器；
  - 也可以设置为类似于 Google Chrome等值；
- **compress是否为静态文件开启gzip compression：**
  - 默认值是false，可以设置为true；

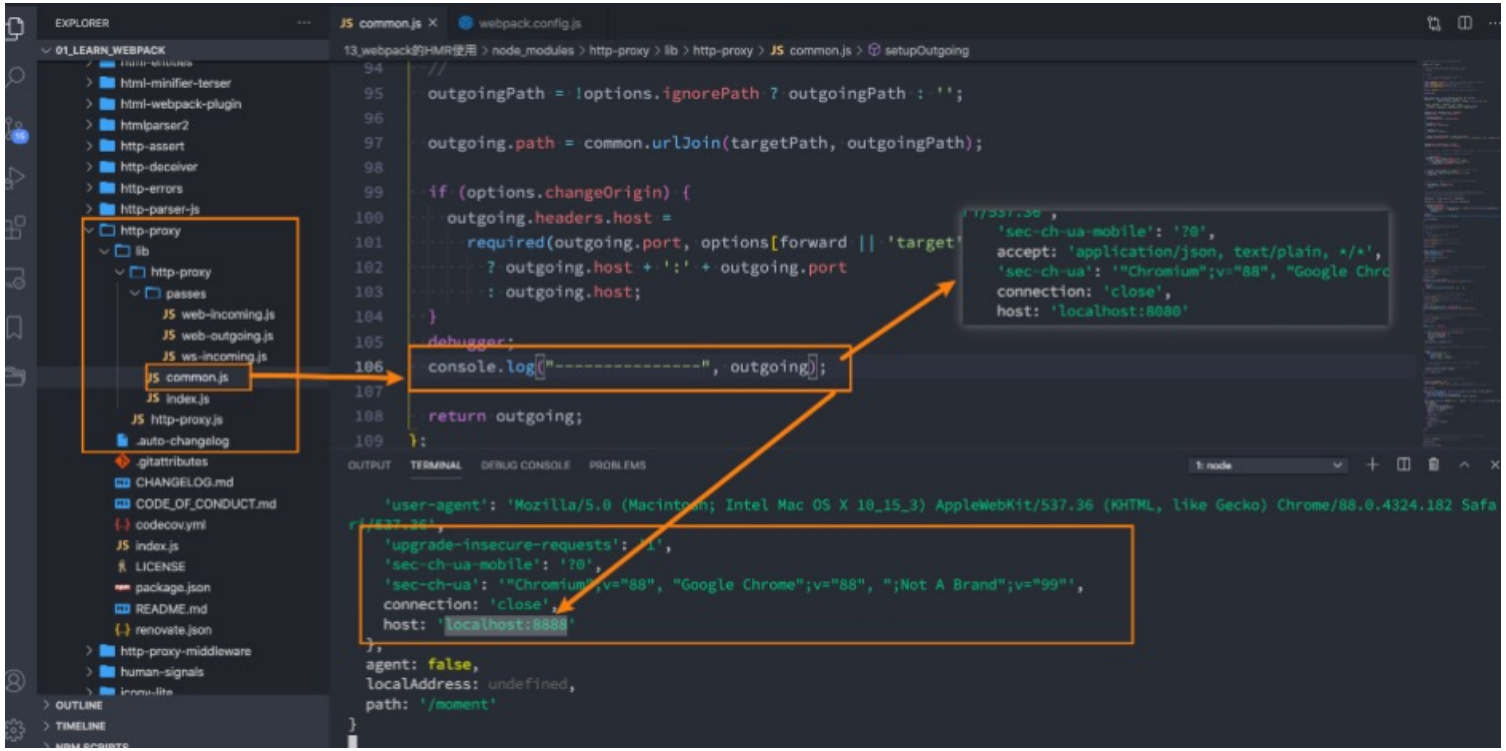


Proxy

- **proxy是我们开发中非常常用的一个配置选项，它的目的设置代理来解决跨域访问的问题：**
  - 比如我们的一个api请求是 http://localhost:8888，但是本地启动服务器的域名是 http://localhost:8000，这个时候发送网络请求就会出现跨域的问题；
  - 那么我们可以将请求先发送到一个代理服务器，代理服务器和API服务器没有跨域的问题，就可以解决我们的跨域问题了；
- **我们可以进行如下的设置：**
  - **target**：表示的是代理到的目标地址，比如 /api-hy/moment会被代理到 http://localhost:8888/api-hy/moment；
  - **pathRewrite**：默认情况下，我们的 /api-hy 也会被写入到URL中，如果希望删除，可以使用pathRewrite；
  - **secure**：默认情况下不接收转发到https的服务器上，如果希望支持，可以设置为false；
  - **changeOrigin**：它表示是否更新代理后请求的headers中host地址；

changeOrigin的解析

- 这个 changeOrigin官方说的非常模糊，通过查看源码我发现其实是要修改代理请求中的headers中的host属性：
  - 因为我们真实的请求，其实是需要通过 http://localhost:8888来请求的；
  - 但是因为使用了代码，默认情况下它的值时 http://localhost:8000；
  - 如果我们需要修改，那么可以将changeOrigin设置为true即可；



historyApiFallback

- **historyApiFallback**是开发中一个非常常见的属性，它主要的作用是解决SPA页面在路由跳转之后，进行页面刷新时，返回404的错误。
- boolean值：默认是false
  - 如果设置为true，那么在刷新时，返回404错误时，会自动返回 index.html 的内容；
- object类型的值，可以配置rewrites属性（了解）：
  - 可以配置from来匹配路径，决定要跳转到哪一个页面；

- 事实上devServer中实现historyApiFallback功能是通过connect-history-api-fallback库的：
  - 可以查看connect-history-api-fallback 文档

resolve模块解析

- resolve用于设置模块如何被解析：
  - 在开发中我们会有各种各样的模块依赖，这些模块可能来自于自己编写的代码，也可能来自第三方库；
  - resolve可以帮助webpack从每个 require/import 语句中，找到需要引入到合适的模块代码；
  - webpack 使用 enhanced-resolve 来解析文件路径；
- **webpack能解析三种文件路径：**
- 绝对路径
  - 由于已经获得文件的绝对路径，因此不需要再做进一步解析。
- 相对路径
  - 在这种情况下，使用 import 或 require 的资源文件所处的目录，被认为是上下文目录；
  - 在 import/require 中给定的相对路径，会拼接此上下文路径，来生成模块的绝对路径；
- 模块路径
  - 在 resolve.modules中指定的所有目录检索模块；
    - ✓ 默认值是 ['node_modules']，所以默认会从node_modules中查找文件；
  - 我们可以通过设置别名的方式来替换初识模块路径，具体后面讲解alias的配置；

确实文件还是文件夹

- 如果是一个文件：
  - 如果文件具有扩展名，则直接打包文件；
  - 否则，将使用 resolve.extensions选项作为文件扩展名解析；
- 如果是一个文件夹：
  - 会在文件夹中根据 resolve.mainFiles配置选项中指定的文件顺序查找；
    - ✓ resolve.mainFiles的默认值是 ['index']；
    - ✓ 再根据 resolve.extensions来解析扩展名；

extensions和alias配置

- extensions是解析到文件时自动添加扩展名：
  - 默认值是 ['.wasm', '.mjs', '.js', '.json']；
  - 所以如果我们代码中想要添加加载 .vue 或者 jsx 或者 ts 等文件时，我们必须自己写上扩展名；
- 另一个非常好用的功能是配置别名alias：
  - 特别是当我们项目的目录结构比较深的时候，或者一个文件的路径可能需要 ../../../这种路径片段；
  - 我们可以给某些常见的路径起一个别名；

```
resolve: {
  extensions: [".wasm", ".mjs", ".js", ".json", ".jsx", ".ts", ".vue"],
  alias: {
    "@": resolveApp("./src"),
    pages: resolveApp("./src/pages"),
  },
},
```

如何区分开发环境

- 目前我们所有的webpack配置信息都是放到一个配置文件中的：webpack.config.js
  - 当配置越来越多时，这个文件会变得越来越不容易维护；
  - 并且某些配置是在开发环境需要使用的，某些配置是在生成环境需要使用的，当然某些配置是在开发和生成环境都会使用的；
  - 所以，我们最好对配置进行划分，方便我们维护和管理；
- 那么，在启动时如何可以区分不同的配置呢？
  - 方案一：编写两个不同的配置文件，开发和生成时，分别加载不同的配置文件即可；
  - 方式二：使用相同的一个入口配置文件，通过设置参数来区分它们；

```
"scripts": {
  "build": "webpack --config ./config/common.config --env production",
  "serve": "webpack serve --config ./config/common.config"
},
```

入口文件解析

- 我们之前编写入口文件的规则是这样的：./src/index.js，但是如果我们的配置文件所在的位置变成了 config 目录，我们是否应该变成 ../src/index.js呢？
  - 如果我们这样编写，会发现是报错的，依然要写成 ./src/index.js；
  - 这是因为入口文件其实是和另一个属性时有关的 context；
- context的作用是用于解析入口（entry point）和加载器（loader）：
  - 官方说法：默认是当前路径（但是经过我测试，默认应该是webpack的启动目录）
  - 另外推荐在配置中传入一个值；

```
// context是配置文件所在目录
module.exports = {
  context: path.resolve(__dirname, "./"),
  entry: "../src/index.js"
}
```

```
// context是上一个目录
module.exports = {
  context: path.resolve(__dirname, "../"),
  entry: "./src/index.js"
}
```

区分开发和生成环境配置

- 这里我们创建三个文件：
  - webpack.comm.conf.js
  - webpack.dev.conf.js
  - webpack.prod.conf.js

- 具体的分离代码这里不再给出，查看课堂代码；

Vue CLI脚手架

- 什么是Vue脚手架？
  - 我们前面学习了如何通过webpack配置Vue的开发环境，但是在真实开发中我们不可能每一个项目从头来完成所有的webpack配置，这样显示开发的效率会大大的降低；
  - 所以在真实开发中，我们通常会使用脚手架来创建一个项目，Vue的项目我们使用的就是Vue的脚手架；
  - 脚手架其实是建筑工程中的一个概念，在我们软件工程中也会将一些帮助我们搭建项目的工具称之为脚手架；

- Vue的脚手架就是Vue CLI：
  - CLI是Command-Line Interface, 翻译为命令行界面；
  - 我们可以通过CLI选择项目的配置和创建出我们的项目；
  - Vue CLI已经内置了webpack相关的配置，我们不需要从零来配置；

Vue CLI 安装和使用

- 安装Vue CLI（目前最新的版本是v4.5.13）
  - 我们是进行全局安装，这样在任何时候都可以通过vue的命令来创建项目；

```
npm install @vue/cli -g
```

- 升级Vue CLI：
  - 如果是比较旧的版本，可以通过下面的命令来升级

```
npm update @vue/cli -g
```

- 通过Vue的命令来创建项目

```
Vue create 项目的名称
```

vue create 项目的过程

```
Vue CLI v4.5.13          made by coderwhy
? Please pick a preset: (Use arrow keys)
❯ Default ([Vue 2] babel, eslint)  选择vue2的版本，并且默认选择babel、eslint
  Default (Vue 3) ([Vue 3] babel, eslint)  选择vue3的版本，并且babel、eslint
  Manually select features  手动来选择希望获取到的特性
```

选择预设

```
? Please pick a preset: Manually select features
? Check the features needed for your project: (Press <space> to select, <a> to toggle all, <i> to invert selection)
❯◉ Choose Vue version  是否选择vue的版本，默认现在是vue2
 ◉ Babel  是否选择babel
 ○ TypeScript 是否使用TypeScript
 ○ Progressive Web App (PWA) Support  项目是否支持PWA
 ○ Router 是否默认添加Router路由
 ○ Vuex  是否默认添加Vuex状态管理
 ○ CSS Pre-processors 是否选择CSS预处理器
 ◉ Linter / Formatter 是否选择ESLint对代码进行格式化限制
 ○ Unit Testing 是否添加单元测试
 ○ E2E Testing  是否添加E2E测试
```

选择需要的特性

made by coderwhy

```
? Choose a version of Vue.js that you want to start the project with (Use arrow keys)
❯ 2.x
  3.x
```

选择Vue的版本

made by coderwhy

```
? Where do you prefer placing config for Babel, ESLint, etc.? (Use arrow keys)
❯ In dedicated config files
  In package.json
```

是否将配置信息放到独立的文件中

made by coderwhy

项目的目录结构


```
"scripts": {
  "serve": "vue-cli-service serve",
  "build": "vue-cli-service build"
},
```

Vue CLI的运行原理