学习 Webpack5 之路（基础篇）

上传日期：2021.08.02 1137

本文从 webpack 是什么、为什么选择 webpack、webpack的基本概念介绍 3个角度进行讲述，从 Webpack 基础着手，和你一起了解 webpack。

前言

对我来说，掌握一个工具最好的方式，就是在学习的过程中，总结并记录，尝试把自己学到的东西进行表达并分享，在分享的过程中，找到一个个同行的小伙伴，一起交流、学习，感受到学习技术的简单和纯粹。

《学习 Webpack5 之路》系列文章，标题灵感来自朴树的《平凡之路》，一直渴望一场走到东极岛的公路旅行，或许学习之路也能和平凡之路一样，让我获得满足和快乐。

《学习 Webpack5 之路》系列文章将分为以下 4 个系列，敬请期待：

- 基础篇
- 实践篇
- 优化篇
- 原理篇

本文依赖的 webpack 版本信息如下：

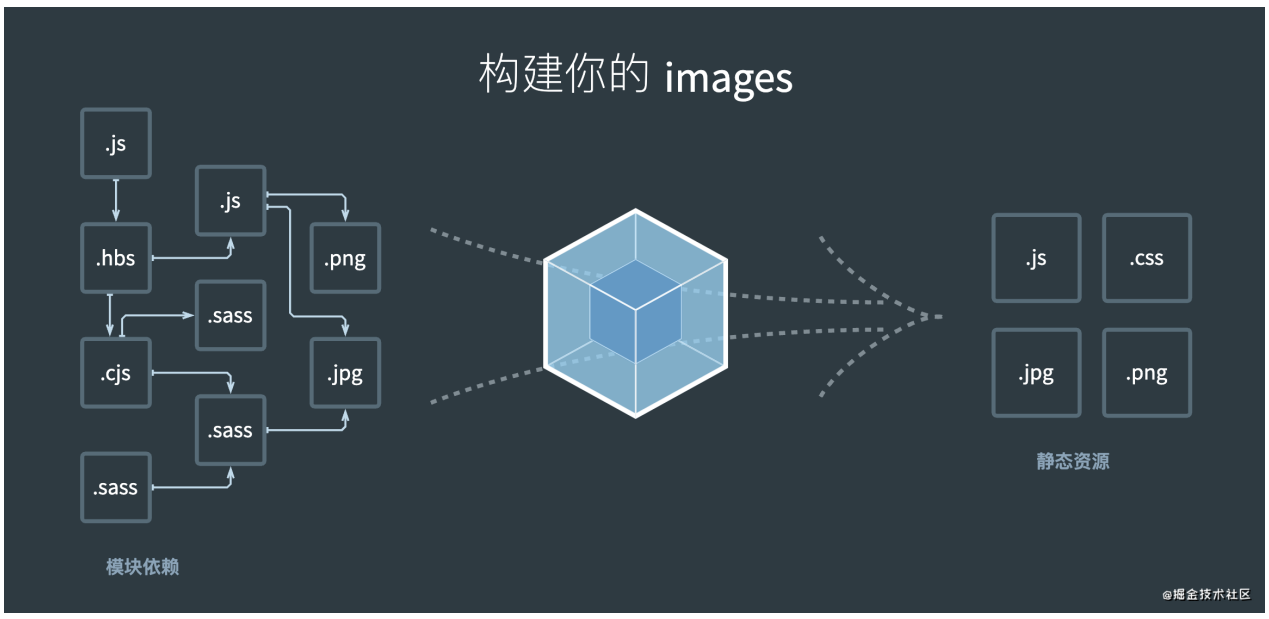
- webpack-cli@4.7.2
- webpack@5.46.0

一、Webpack 是什么

引入 webpack 官网 介绍：

> 本质上，*webpack* 是一个现代 *JavaScript* 应用程序的静态模块打包器*(module bundler)* 。当 *webpack* 处理应用程序时，它会递归地构建一个依赖关系图*(dependency graph)* ，其中包含应用程序需要的每个模块，然后将所有这些模块打包成一个或多个 *bundle*。

webpack 官网 图：



在图中我们可以看到，webpack 将左侧错综复杂的各自不同类型文件的模板依赖关系，包括 .js、.hbs、.cjs、.sass、.jpg、.png 等类型文件，打包成 .js、.css、.jpg、.png 4 种类型的静态资源。

简单来说，webpack 就是一个静态资源打包工具，负责将项目中依赖的各个模块，打包成一个或多个文件。

二、为什么选择 Webpack

> 本文不进行其他打包工具和 *webpack* 的优劣对比，仅介绍 *webpack* 能为开发者做的工作。

1. 模块化开发

在没有各个 webpack 搭建的脚手架（create-react-app、vue-cli 等等）之前，我们通过在 HTML5 文件里引入一个个 Javascript 文件来进行开发，这就可能导致并行请求数量过多、存在重复代码等问题。

而通过 webpack，**我们可以使用 import、require 来进行模块化开发。**

在 webpack 中一切皆模块，js、css、图片、字体都是模块，而且支持静态解析、按需打包、动态加载、代码分离等功能，帮助我们优化代码，提升性能。

```
import { Hello } from './hello.js'
import './assets/style.css'
import MyImage './assets/img.jpg'
```

2. 新语法

Javascript、CSS 的语法规范在不断更新，但是浏览器的兼容性却不能同步的更新，开发者可以通过 webpack 预处理器进行编译，**自由的使用 JS、CSS 等语言的新语法。**

webpack 使用 loader 对文件进行预处理。你可以构建包括 JavaScript 在内的任何静态资源，如 Less、Sass、ES6、TypeScript。

通过预处理器将 TypeScript 编译成 JavaScript、SCSS 编译成 CSS、ES6 编译成 ES5 等。

开发者还可以使用 Node.js 轻松编写自己的 loader。

常用预处理器：

- `babel-loader` 使用 Babel 加载 ES2015+ 代码并将其转换为 ES5；
- `less-loader` 加载并编译 LESS 文件；
- `sass-loader` 加载并编译 SASS/SCSS 文件；
- `postcss-loader` 使用 PostCSS 加载并转换 CSS/SSS 文件。

3. 主流框架脚手架

Vue 脚手架 `vue-cli`、React 脚手架 `creact-react-app`、Taro 脚手架 `taro-cli` 都是使用 webpack，开发者掌握 webpack 后，可以自由配置脚手架，根据项目需要，调整 webpack 配置，以提高项目性能。

4. 其他

webpack 除了让开发者能够拥有**【模块化开发+新语言+新框架】**的开发体验。

还有**以下优点**：

- 拥有依赖管理、动态打包、代码分离、按需加载、代码压缩、静态资源压缩、缓存等配置；
- webpack 扩展性强，插件机制完善，开发者可自定义插件、loader；
- webpack 社区庞大，更新速度快，轮子丰富；

如使用 ant-design 搭建的中后台项目，ant-desgin 提供了 webpack 定制主题的相关文档，较其他打包工具定制起来就简单很多，易上手。

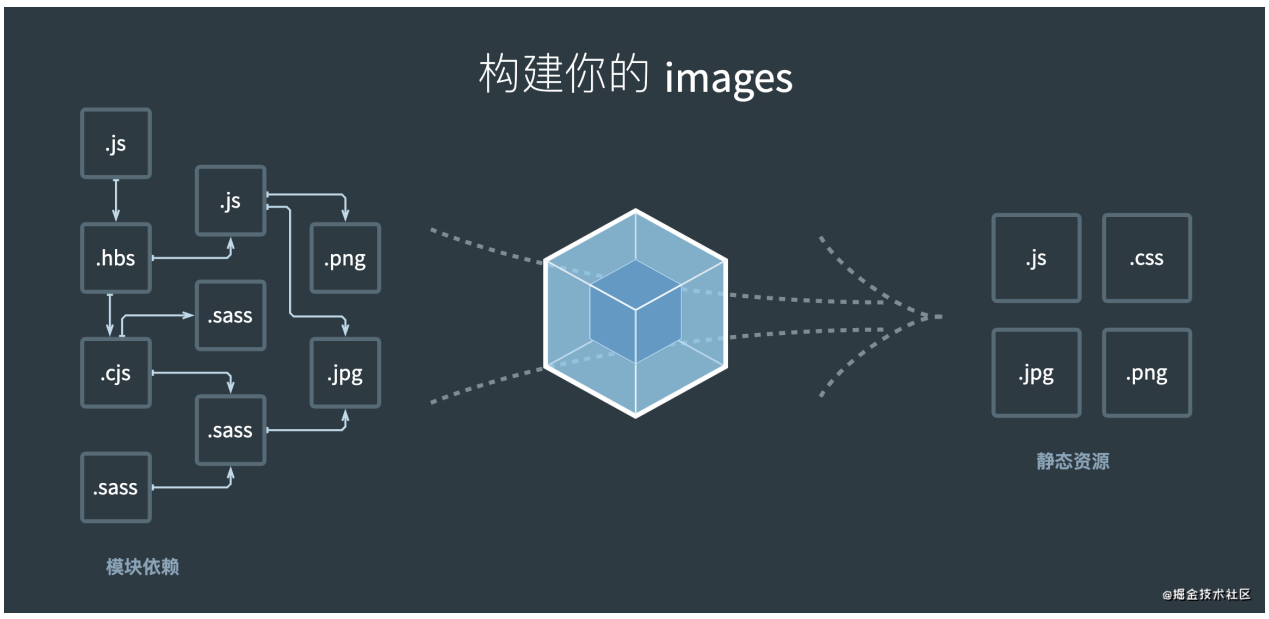
因为 webpack 的这些优点，大部分的大型项目会选择 webpack 进行项目构建。

三、Webpack 的基本概念介绍

1. dependency graph（依赖图）

上文有提到，当 webpack 处理应用程序时，它会递归地构建一个依赖关系图*(dependency graph)*，那么依赖关系图是什么呢?

依赖图指的就是文件和文件直接的依赖关系，如上文引入过的图：



webpack 通过依赖关系图可以获取非代码资源，如 images 或 web 字体等。并会把它们作为 依赖 提供给应用程序。

每个模块都可以**明确表述它自身的依赖**，在打包时可根据依赖进行打包，避免打包未使用的模块。

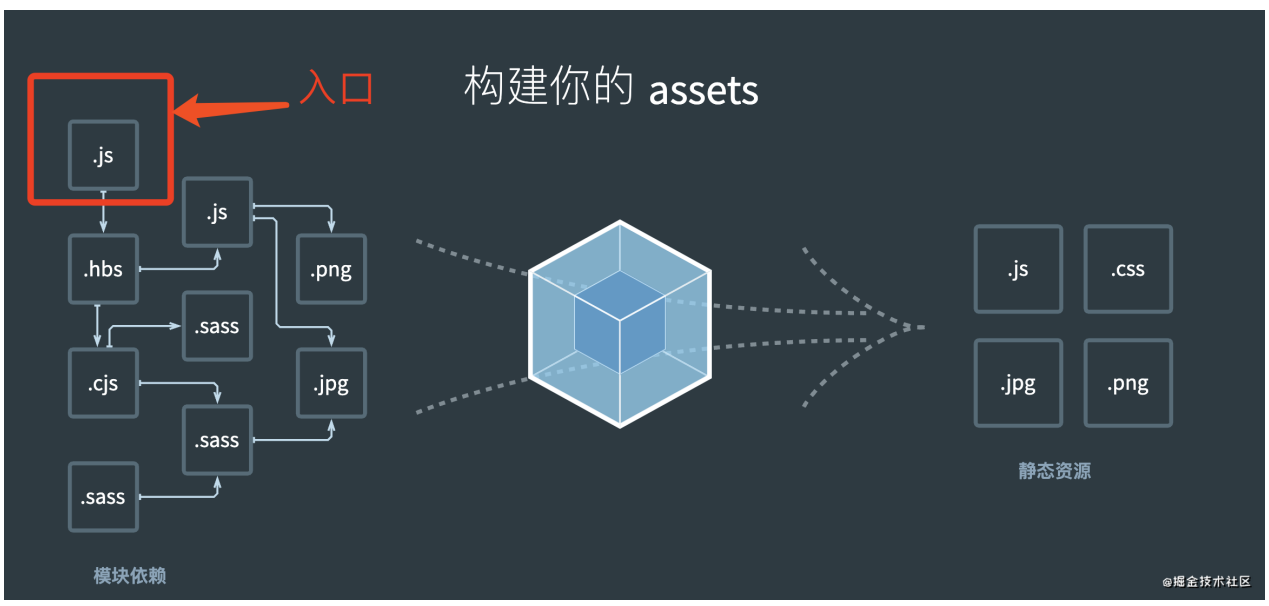
2. entry（入口）

入口是指依赖关系图的开始，从入口开始寻找依赖，打包构建。

webpack 允许一个或多个入口配置。

配置示例如下：

```
module.exports = {
  entry: 'index.js',
};
```

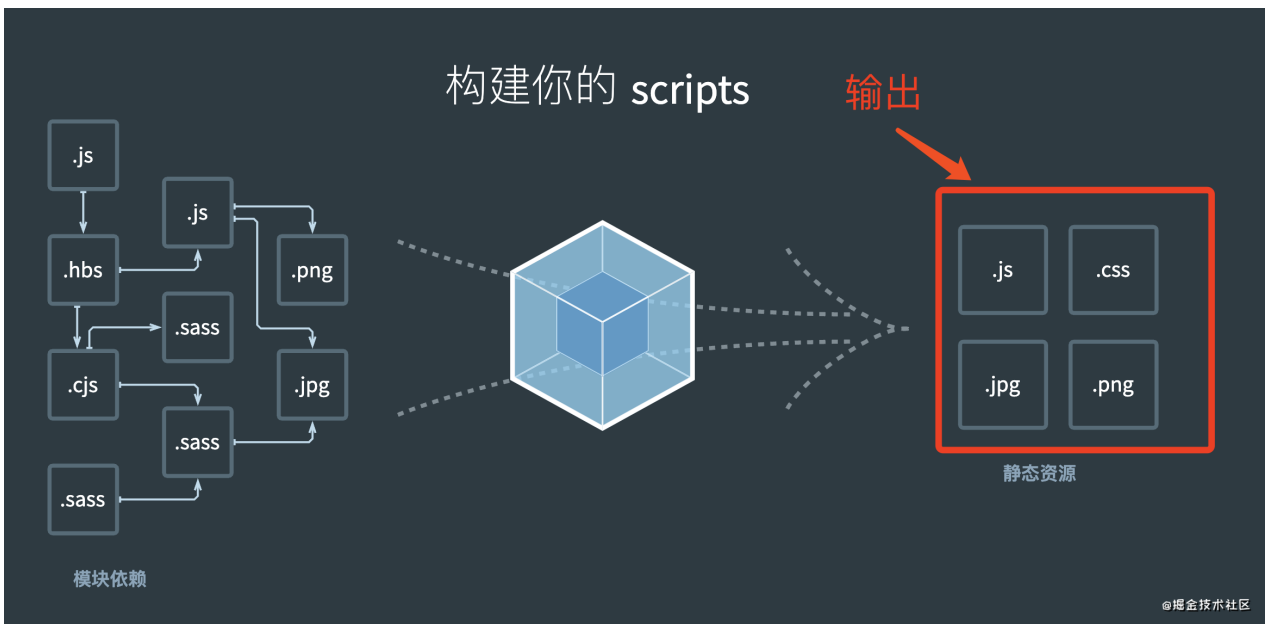


3. output（输出）

输出则是用于配置 webpack 构建打包的出口，如打包的位置，打包的文件名等等。

配置示例如下：

```
module.exports = {
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'my-first-webpack.bundle.js',
  },
};
```



4. loader

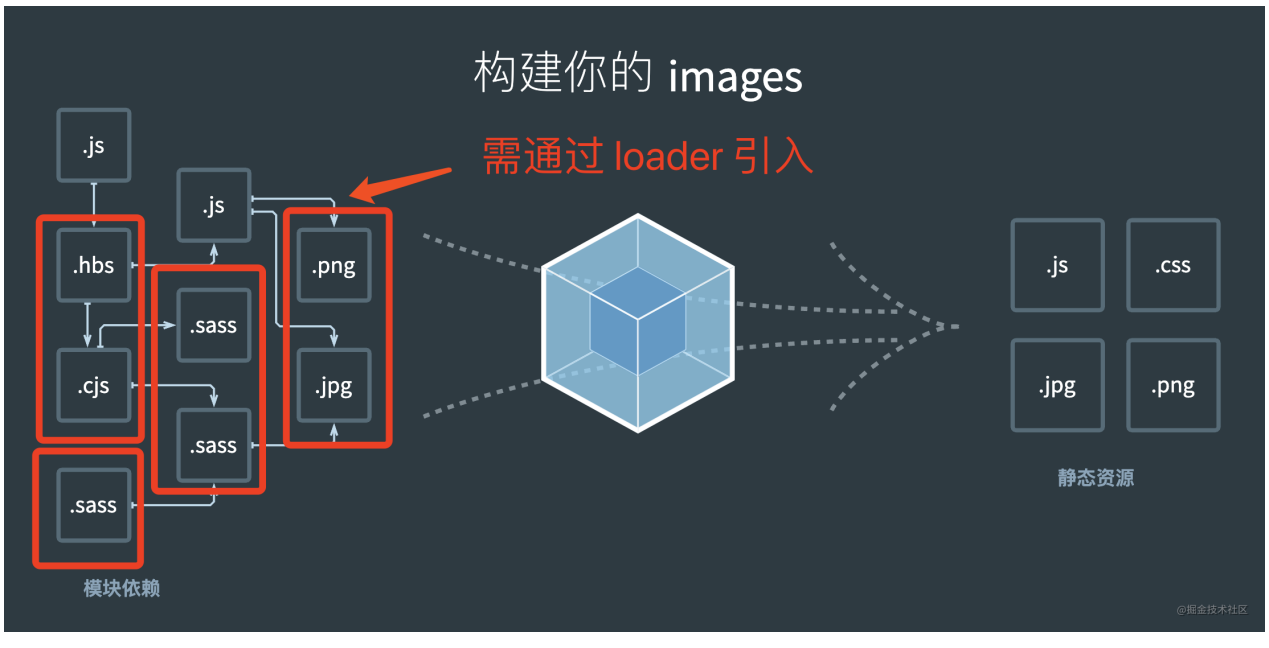
webpack 自带 JavaScript 和 JSON 文件的打包构建能力，无需格外配置。

而其他类型的文件，如 CSS、TypeScript，则需要安装 loader 来进行处理。

loader 让 webpack 能够去处理其他类型的文件，并将它们转换为有效 模块，以供应用程序使用，以及被添加到依赖图中。

配置示例如下：

```
module.exports = {
  module: {
    rules: [{ test: /.txt$/, use: 'raw-loader' }],
  },
};
```



5. plugin（插件）

插件则是用于扩展 webpack 的能力，常见的插件有：

- ProgressBarPlugin：编译进度条；
- BundleAnalyzerPlugin：打包体积分析；
- MiniCssExtractPlugin：提取 CSS 到独立 bundle 文件。

配置示例如下：

```
const HtmlWebpackPlugin = require('html-webpack-plugin'); // 通过 npm 安装

module.exports = {
  plugins: [new HtmlWebpackPlugin({ template: './src/index.html' })],
};
```

插件丰富，开发者社区同样提供了大量插件，也使得 webpack 的可用功能更加多样。

6. mode（模式）

webpack5 提供了模式选择，包括开发模式、生产模式、空模式，并对不同模式做了对应的内置优化。可通过配置模式让项目性能更优。

配置示例如下：

```
module.exports = {
  mode: 'development',
};
```

7. resolve（解析）

resolve 用于设置模块如何解析，常用配置如下：

- alias：配置别名，简化模块引入；
- extensions：在引入模块时可不带后缀；
- symlinks：用于配置 npm link 是否生效，禁用可提升编译速度。

配置示例如下：

```
module.exports = {
    resolve: {
        extensions: ['.js', '.jsx', '.ts', '.tsx', '.json', '.d.ts'],
        alias: {
          '@': paths.appSrc,
        },
        symlinks: false,
      }
}
```

8. optimization（优化）

optimization 用于自定义 webpack 的内置优化配置，一般用于生产模式提升性能，常用配置项如下：

- minimize：是否需要压缩 bundle；
- minimizer：配置压缩工具，如 TerserPlugin、OptimizeCSSAssetsPlugin；
- splitChunks：拆分 bundle；
- runtimeChunk：是否需要将所有生成 chunk 之间共享的运行时文件拆分出来。

配置示例如下：

```
module.exports = {
  optimization: {
    minimizer: [
      // 在 webpack@5 中，你可以使用 `...` 语法来扩展现有的 minimizer（即 `terser-webpack-plugin`），将下一行取消注释
      // `...`,
      new CssMinimizerPlugin(),
    ],
    splitChunks: {
      // include all types of chunks
      chunks: 'all',
      // 重复打包问题
      cacheGroups:{
        vendors:{ //node_modules里的代码
          test: /[\\/]node_modules[\\/]/,
          chunks: "all",
          name: 'vendors', //chunks name
          priority: 10, //优先级
          enforce: true
        }
      }
    },
  },
}
```

以上对 webpack 的基本概念做了简单的介绍，为后续实践篇做准备。

四、总结

本文从 webpack 是什么、为什么选择 webpack、webpack的基本概念介绍 3个角度进行讲述，从 Webpack 基础着手，和你一起了解 webpack。

《学习 Webpack5 之路》系列文章将分为以下 4 个系列：

- 基础篇
- 实践篇
- 优化篇
- 原理篇

下一篇《学习 Webpack5 之路（实践篇）》将从实践出发，一起完成一个比较完整的 webpack 配置，敬请期待。

本文源码： webpack Demo0

希望能对你有所帮助，感谢阅读～

别忘了点个赞鼓励一下我哦，笔芯♡

参考资料

- https://webpack.docschina.org/