基本概念

在开始前，先了解几个webpack常用术语的概念。

入口 entry

入口，即为我们项目的入口文件，webpack会从我们指定的入口文件开始，递归查询项目中所有依赖的库、文件。从而生成一张从入口文件开始，各模块之间相互依赖的依赖图。

Webpack默认值的入口是 `src/index.js`，也就是我们上节零配置启动webpack时，默认设定的入口。 可以通过在配置文件中配置`entry`属性，来指定一个（或多个）不同的入口起点。例如：

```
// webpack.config.js
module.exports = {
  entry: './path/to/my/entry/file.js', // 单个入口文件路径
};
```

多entry的配置方式在后续会详细说明。

输出 output

`output`属性告诉webpack，构建完产生的文件要放到哪里，以及如何命名。默认的输出目录为`dist`，默认生成的文件为`main.js`。

可以通过在配置中指定`output`字段，来配置输出：

```
// webpack.config.js
const path = require('path');

module.exports = {
  entry: './path/to/my/entry/file.js',
  output: {
    path: path.resolve(__dirname, 'dist'), // 输出目录
    filename: 'my-output-filename.js', // 输出文件名
  },
};
```

- `output.path` 为输出路径，即构建完的文件要存储到哪个目录。
- `output.filename` 为生成的文件名。

最终生成的构建产物存储在 `[path]/[filename]`。如以上配置产生的文件即为`dist/my-output-filename.js`。

loader

> loader 用于对模块的源代码进行转换

webpack 默认只能理解 JavaScript 和 JSON 文件。在我们实际的开发过程中，我们的js文件中会引入各种类型的文件，如样式相关的 `.css` `.less` `.sass` `.stylus` 等文件，还有js超集`.ts`文件、react的`.jsx`文件，vue的`.vue`文件，还有图片、字体...等各种各样的文件，webpack在遇到这些文件时，并不知道该如何处理这些文件。loader就是用来告诉webpack，如何去处理其他非js类型的文件。

假设我们的入口文件引入了样式文件`index.less`。

这时候我们执行`npm run dev`会报错，因为webpack无法处理`less`类型的文件。 这时候我们只需要配置loader，告诉webpack如何处理即可：

首先安装三个loader：

```
npm i style-loader css-loader less-loader less -D
```

安装 `less` 是因为`less-loader`解析`less`文件时需要依赖`less`库

接着配置loader：

```
// webpack.config.js
module.exports = {
  module: {
    rules: [
      {
        test: /\.less$/i, // 匹配.less后缀结尾的文件
        loader: [
          'style-loader',
          'css-loader',
          'less-loader',
        ],
      },
    ],
  },
};
```

- `module.rules`是一个数组，允许配置多个规则，这些规则从上往下依次执行
- `test`以正则方式匹配我们要要处理的文件，通常我们通过后缀来命中要处理的文件
- loader可以是一个数组，数组中的loader按照从右往左（从下往上）的顺序依次执行

上述示例中，我们设置了规则，当遇到.less结尾的文件时，使用三个loader对其进行处理。注意loader的执行顺序至关重要，一定要按顺序执行：

1. 首先`less-loader`将less语法转换为原生的css语法
2. 接着`css-loader` 的作用是分析不同css文件间的关系，如处理css中的`@import`和`url`这样的外部资源
3. 转换完的css怎么用呢？最后`style-loader`将`css-loader`转换完的css数据，以`style`标签的形式，插入到HTML页面的`head`标签中，这样css才能生效。

以上就是我们对于less文件的常规处理处理方式，类似的我们还可以处理其他类型的文件。至于所不同文件需要的loader，我们可以去webpack社区寻找。后面在实战环节我们会将项目开发中常用的loader都罗列出来。

插件 plugin

> 插件 是 webpack 的 支柱 功能。插件目的在于解决 loader 无法实现的其他事。

loader用于各种类型的文件模块的转换，除此之外我们想要实现其他的功能就要依赖插件plugin来执行了，包括打包优化、资源管理等等。比如我们上面将less文件使用loader处理后，最终以style的形式插入到了HTML页面。如果我们不想让样式以style标签的形式插入到head标签中，而是把css单独抽离出来成一个文件，这种涉及到文件/资源生成的操作，就需要插件来处理。

插件的使用也比较简单，安装插件后只要将插件引入并在配置中声明即可。以将css样式抽离为单独的文件为例，我们使用 `mini-css-extract-plugin`这个插件即可。

首先安装插件`npm i mini-css-extract-plugin -D`

然后在配置wenpack

```
// webpack.config.js
const MiniCssExtractPlugin = require("mini-css-extract-plugin");

module.exports = {
  plugins: [new MiniCssExtractPlugin()], // 使用plugin
  module: {
    rules: [
      {
        test: /\.less$/i,
        loader: [
          MiniCssExtractPlugin.loader,
          'style-loader',
          'css-loader',
          'less-loader',
        ],
      },
    ],
  },
};
```

这样当我们再次执行`npm run build`就会发现css已经被打包成一个单独的css文件了。

这里只是举例来说明插件的使用方法，后续会在实战章节介绍其他常用的插件。

模式 mode

模式分为 `development`, `production` 或 `none` 之中的一个，其默认值为 `production`。

- `development`即为本地开发模式，在本地开发环节使用。会将 `DefinePlugin` 中 `process.env.NODE_ENV` 的值设置为 `development`。启用部分插件以优化开发环境。
- `production`为生产模式，在我们编译代码上线时使用。会将 `DefinePlugin` 中 `process.env.NODE_ENV` 的值设置为 `production`。启用部分插件以优化打包产物，如：
  - FlagDependencyUsagePlugin 删除无用代码
  - FlagIncludedChunksPlugin 删除无用代码

    - ModuleConcatenationPlugin 作用域提升
    - NoEmitOnErrorsPlugin 编译出现错误，跳过输出阶段
    - TerserPlugin js压缩
- `none` 不使用任何默认优化选项

注：`DefinePlugin`是webpack5自带的一个plugin，可以用来定义webpack运行时的环境变量。

配置方式如下：

```
// webpack.config.js
module.exports = {
  mode: 'production',
};
```

`mode`我们一般会根据根据不同的场景动态设置为`development`和`production`两者中的一个，在后续章节会有具体案例。