概念

webpack

- 用于现在JavaScript应用程序的静态模块打包工具
- 处理时，从内部的一个或多个入口构建一个依赖图(dependency graph)
- 然后将需要的每个模块组合成一个或者多个bundles，均为静态资源

一些核心概念

入口(entry)

- 入口起点只是用那个模块作为构建内部依赖图的起点
- 进入起点后，就会去找出有哪些模块和库时入口起点直接/间接依赖的
- 默认是./src/index.js
- 可以在webpack.config.js中进行配置

输出(output)

- 对于每个入口在哪里输出所创建的bundle
- 主要输出文件的默认值是./dist/main.js
- 所有的生成文件都默认放在./dist文件夹中
- 在配置中匹配指定的output字段
- output
  - path: path.resolve(__dirname, "dist"),
  - filename: [filename],
- 这里的path是一个Node.js核心模块，用于操作文件路径

loader

- webpack自带有理解JavaScript和JSON文件的能力
- loader让它能够处理其他类型的文件，并转换为有效的模块
- 共应用程序使用
- 并且被添加带依赖图中
- webpack能够通过import导入任何类型的模块，loader让它有处理它们的能力
- 配置中loader有两个属性
  - test属性，正则，识别那些文件会被转换
  - use属性，定义在转换时，对这些文件类型使用哪个loader

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

- 使用一个单独的module来包括一个rules属性，对应一堆转换的规则，每个规则包括loader的两个属性，如
  - 表示每次碰到require() / import语句中后缀为'.txt'的路径，先使用raw-loader进行转换
  - 在配置rules要定义在module.rules，而不是直接定义在rules属性
  - 同时test属性对应一个正则对象，不用任何引号

插件(plugin)

- loader用于转换某些类型的模块
- 而插件用于执行范围更广的任务，包括
  - 打包优化
  - 资源管理
  - 诸如环境变量
- 要使用一个插件
  - 首先通过require函数引入它
  - 然后将其添加到plugins数组中
  - 多数插件需要使用一个option自定义，再使用new创建实例对象传入option
  - 如果需要因为不同目的多次使用同一个插件，这时候要放到plugins数组中的就是一个插件实例

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

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

- 如
  - 注意，这里的plugins数组和rules不同，是一个顶级属性
  - 上面创建了一个html-webpack-plugin的实例对象，并且为应用程序生成一个HTML文件，自动将生成的所有bundle都会注入到此文件中

模式(mode)

- 选择三者之一来设置mode参数
  - development
  - production
  - none
- 默认为production

浏览器兼容性(broser compatibility)

- 支持所有符合ES5标准的浏览器 除了IE8及以下版本
- webpack的import()动态加载语句以及require.ensure()需要Promise
- 如果想要支持旧版本浏览器，提前加载polyfill

环境(environtment)

- Webpack 5运行于Node.js v10.13.0+