loader

url-loder

> 如果使用url-loader,会将图片处理成base64文件,并返回一个data-url打包到bundle.js中,并不会单独生成一个文件，这虽然会省去http请求，但如果图片文件过大，页面加载会很慢，如果图片过小，这种加载方法会少一次http请求

file-loader

> 使用file-loader,会将文件上的import/require解析为url,发送到输出文件夹并返回相对url

entry

> 单页应用可传入字符串，多页应用可传入对象

```
    entry:'./src/index.js',
    entry:{
        //将src下的index.js打包两份
        main:'./src/index.js',
        sub:'./src/index.js',
    },
```

output

```
 output:{
        //publicPath:'http://cdn.com.cn',//将静态资源放在cdn上
        //filename:'bundle.js',//打包后的文件名，默认是main.js
        filename:'[name].js',//对应entry的名称
        path:path.resolve(__dirname,'dist'),//打包后的文件要放在哪，后面是绝对路径，打包后生成dist文件夹
    },
```

plugin

> 可以在webpack运行到某个时刻的时候，帮你做一些事情，比如HtmlWebpackPlugin会在打包结束后生成一个html,并将打包后的js注入html文件

```
    plugins:[
//此插件会在打包结束后，自动生成一个html文件，并把打包生成的js自动引入到这个html文件中
                new HtmlWebpackPlugin({
                    template:'src/index.html'
                }),
//clean-webpack-plugin:在打包之前会先删除dist下的内容，然后再生成
                new CleanWebpackPlugin()
            ],
```

sourceMap

> 是一种映射关系。他知道dist目录下main.js96行报错，实际上对应的是src目录下的index.js文件中的第一行报错，使用映射关系，可配置devtool,生产环境配置成cheap-module-source-map比较好

```
devtool:'cheap-module-eval-source-map',//将map文件以base64字符串的形式打包到对应的脚本的最末尾
```

mode

> 默认production,development||none,设置为development时，生成的bundle.js是不被压缩的代码

```
mode:'development'
```

自动重新打包的几种方式

> 修改package.json，监听打包文件，若有变化，自动重新打包，而不用手动重新打包，用以下几种方式

命令行监听

```
//命令中添加--watch参数
"scripts": {
    "watch": "webpack --watch"
  },
```

webpack-dev-server

> npm install webpack-dev-server,然后再webpack配置文件进行配置，打包后，会帮助启动一个http服务，文件内容变化会重新打包并刷新页面

```
//package.json配置
"scripts": {
    "watch": "webpack --watch"
},
//webpack.config.js
module.exports={
    mode:'development',
    devtool:'cheap-module-eval-source-map',
    entry:'./src/index.js',
    devServer:{
        open:true
    },
}
```

使用node中间件打包

> 在package.json中添加script命令，然后写一个node服务脚本

```
//package.json
 "scripts": {
    "middleware": "node server.js"
  },
--------------------------------------------------------------------------------
//根目录下添加server.js
const express=require('express');
const webpack= require('webpack');
const webpackMiddleware=require('webpack-dev-middleware');
//引入webpack的配置文件
const config=require('./webpack.config.js');
//node下webpack的api,用webpack结合config,编译文件
const complier=webpack(config);
//服务器实例
const app=express();
//使用中间件,只要文件变化，compiler就会重新编译，打包输出的路径就是config文件下的output的
publicPath
app.use(webpackMiddleware(complier,{
   // publicPath:config.output.publicPath
}));
app.listen(8080,()=>{
    console.log('server is running');
});
```