首先需要node环境

详情参考(https://juejin.im/post/5adea0106fb9a07a9d6ff6de#heading-9)

安装：

1 如果是全局安装 npm i -g webpack webpack-cli 在之后使用打包命令直接使用 webpack

如果是局部安装 npm i -D webpack webpack-cli 在之后使用中 npx webpack

2 webpack命令打包（0配置）

npx webpack // 不设置mode的情况下 打包出来的文件自动压缩

npx webpack --mode development // 设置mode为开发模式，打包后的文件不被压缩

打包完之后会在根目录生成一个dist目录并存在一个打包好的main.js文件.

3 创建webpack.config.js

```
    plugins: [
        // 通过new一下这个类来使用插件
        new HtmlWebpackPlugin({
            // 用哪个html作为模板
            // 在src目录下创建一个index.html页面当做模板来用
            template: './src/index.html',
            hash: true, // 会在打包好的bundle.js后面加上hash串
        })
    ]
```

4 启动devServer需要安装 webpack-dev-server

npm i webpack-dev-server -D

5 配置package.json文件

```
"scripts": {
    "build": "webpack",
    "dev": "webpack-dev-server"
},
```

npm run build就是我们打包后的文件，这是生产环境下，上线需要的文件
npm run dev是我们开发环境下打包的文件，当然由于devServer帮我们把文件放到内存中了，所以并不会输出打包后的dist文件夹

6 多入口文件

```
module.exports = {
    // 1.写成数组的方式就可以打出多入口文件，不过这里打包后的文件都合成了一个
    // entry: ['./src/index.js', './src/login.js'],
    // 2.真正实现多入口和多出口需要写成对象的方式
    entry: {
        index: './src/index.js',
        login: './src/login.js'
    },
    output: {
        // 1. filename: 'bundle.js',
        // 2. [name]就可以将出口文件名和入口文件名一一对应
        filename: '[name].js',      // 打包后会生成index.js和login.js文件
        path: path.resolve('dist')
    }
}
```

7 配置html模板

我们需要实现html打包功能，可以通过一个模板实现打包出引用好路径的html来
这就需要用到一个常用的插件了，**html-webpack-plugin**，
npm i html-webpack-plugin -D

```
    plugins: [
        // 通过new一下这个类来使用插件
        new HtmlWebpackPlugin({
            // 用哪个html作为模板
            // 在src目录下创建一个index.html页面当做模板来用
            template: './src/index.html',
            hash: true, // 会在打包好的bundle.js后面加上hash串
        })
    ]
```

多页面配置的时候

```
    plugins: [
        new HtmlWebpackPlugin({
            template: './src/index.html',
            filename: 'index.html',
            chunks: ['index']   // 对应关系,index.js对应的是index.html
        }),
        new HtmlWebpackPlugin({
            template: './src/login.html',
            filename: 'login.html',
            chunks: ['login']   // 对应关系,login.js对应的是login.html
        })
    ]
```

8 引用css文件

可以在src/index.js里引入css文件，到时候直接打包到生产目录下需要下载一些解析css样式的loader

npm i style-loader css-loader -D

npm i less less-loader -D

```
module: {
    rules: [
        {
            test: /\.css$/,     // 解析css
            use: ['style-loader', 'css-loader'] // 从右向左解析
            /*
                也可以这样写，这种方式方便写一些配置参数
                use: [
                    {loader: 'style-loader'},
                    {loader: 'css-loader'}
                ]
            */
        }
    ]
}
```

拆分css

// @next表示可以支持webpack4版本的插件
npm i extract-text-webpack-plugin@next -D
(还可以使用npm i mini-css-extract-plugin -D)

```
module: {
    rules: [
        {
            test: /\.css$/,
            use: ExtractTextWebpackPlugin.extract({
                // 将css用link的方式引入就不再需要style-loader了
                use: 'css-loader'
            })
        }
    ]
},
plugins: [
    new HtmlWebpackPlugin({
        template: './src/index.html',
    }),
    // 拆分后会把css文件放到dist目录下的css/style.css
    new ExtractTextWebpackPlugin('css/style.css')
]
```

9 引用图片

处理图片也需要loader

如果是在css文件里引入的如背景图之类的图片，就需要指定一下相对路径

npm i url-loader -D

```
rules: [
    {
        test: /\.css$/,
        use: ExtractTextWebpackPlugin.extract({
            use: 'css-loader',
            publicPath: '../'
        })
    },
    {
        test: /\.(jpe?g|png|gif)$/,
        use: [
            {
                loader: 'url-loader',
                options: {
                    limit: 8192,    // 小于8k的图片自动转成base64格式，并且不会存在实体图片
                    outputPath: 'images/'   // 图片打包后存放的目录
                }
            }
        ]
    }
```

页面img引用图片

npm i html-withimg-loader -D

```
module.exports = {
    module: {
        rules: [
            {
                test: /\.(htm|html)$/,
                use: 'html-withimg-loader'
            }
        ]
    }
}
```

引用字体图片和svg图片

npm i file-loader -D

```
module.exports = {
    module: {
        rules: [
            {
                test: /\.(eot|ttf|woff|svg)$/,
                use: 'file-loader'
            }
        ]
    }
}
```

10 添加css3前缀

通过postcss中的autoprefixer可以实现将CSS3中的一些需要兼容写法的属性添加响应的前缀，这样省去我们不少的时间

npm i postcss-loader autoprefixer -D

安装后，我们还需要像webpack一样写一个config的配置文件，在项目根目录下创建一个**postcss.config.js**文件，配置如下

```
module.exports = {
    plugins: [require('autoprefixer')]  // 引用该插件即可了
}
```

然后在webpack里配置postcss-loader

```
module.exports = {
    module: {
        rules: [
            {
                test: /\.css$/,
                use: ['style-loader', 'css-loader', 'postcss-loader']
            }
        ]
    }
}
```

11 转移ES6

Babel会将ES6的代码转成ES5的代码

npm i babel-core babel-loader@7 babel-preset-env babel-preset-stage-0 -D

(npm i @babel/core babel-loader @babel/preset-env -D)

我们可以通过一个.babelrc文件来配置一下，对这些版本的支持

```
{
    "presets": ["env", "stage-0"]   // 从右向左解析
}
```

我们再在webpack里配置一下babel-loader既可以做到代码转成ES5了

```
module.exports = {
    module: {
        rules: [
            {
                test:/\.js$/,
                use: 'babel-loader',
                include: /src/,          // 只转化src目录下的js
                exclude: /node_modules/  // 排除掉node_modules，优化打包速度
            }
        ]
    }
}
```

每次打包之前将dist目录下的文件都清空，然后再把打好包的文件放进去

npm i clean-webpack-plugin -D

```
const { CleanWebpackPlugin } = require("clean-webpack-plugin");

...

plugins: [
    new CleanWebpackPlugin()
]
```

12 启动静态服务器

启动一个静态服务器,默认会自动刷新.

```
module.exports = {
    devServer: {
        contentBase: './dist',
        host: 'localhost',      // 默认是localhost
        port: 3000,             // 端口
        open: true,             // 自动打开浏览器
        hot: true               // 开启热更新
    }
}
```

13 resolve解析

在webpack的配置中，resolve我们常用来配置别名和省略后缀名

```
module.exports = {
    resolve: {
        // 别名
        alias: {
            $: './src/jquery.js'
        },
        // 省略后缀
        extensions: ['.js', '.json', '.css']
    },
}
```

14 提取公共代码

15 指定webpack配置文件

在package.json的脚步里，我们可以配置调用不同的webpack配置文件进行打包，

16 sourcemap

是为了解决开发代码与实际运行代码不一致时帮助我们debug到原始开发代码的技术。