webpack使用心得与记录：

webpack是前端一个工具，可以让各个模块进行加载，预处理，再进行打包，它所有Gunt或Gulp所有进本功能，优点：支持commonJS和AMD模块，吃吃很多模块加载器的调用，可以使模块加载器灵活定制，比如babel-loader加载器，该加载器能使我们使用ES6的语法来编写代码，可以通过配置打包成多个文件，有效的利用浏览器的缓存功提高性能，使用模块加载器，可以支持sass less等处理器进行打包支持静态资源样式以及图片进行打包

使用：

1.安装 mkdir webpack-test

cd webpack-test
npm init
cnpm install webpack —save-dev
(注意还要全局安装 npm install wepack -g)

2.新建文件hell.js

打包 webpack hello.js hello.bundle.js

3.再建立world.js 在hello.js中引入 require(‘hello.js’)

在打包会看到两个编号 0 和 1 在打包完成文件中bundle.js可以看出webpack是根据编号来找到其他文件

webpack 默认不支持打包css文件的打包 需要加入loader

安装npm install css-loader —save-dev
　　npm install style-loader —save-dev

4.新建style.css

引入时需要require(!style-loader!css-loader!./style.css);

也可以不引入直接使用命令

webpack hello.js bundle.js --module-bind 'css=style-loader!css-loader'

5.其他参数 —watch —progress —display-modules 等等

webpack 默认配置文件webpack.config.js

直接输入web pack 会找项目下的web pack.config.js文件

也可以使用—config 参数来执行

6.如果想要使用参数可以搭配npm使用在package.json中的script中加入

```
"scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "webpack": "webpack --config webpack.config.js --progress --display-modules --colors --display-reasons”
  },
```

然后运行npm run webpack

web pack.config.js 文件

```
module.exports = {
　　// 打包源文件
```

```
    entry: './src/script/main.js',
    //生成目标文件
    output: {
        // 生成目标文件目录
        path: './dist/js',
        // 目标文件文件名
        filename: 'bundle.js'
    }

}
```

7.webpack 怎么生成html自动引入js文件 使用插件

web pack 插件使用 html-webpack-plugin
安装 cnpm install html-webpack-plugin —save-dev
引入：

```
var htmlWebpackPlugin = require('html-webpack-plugin');
```

在web pack.config.js中加入

```
plugins: [
        new htmlWebpackPlugin({
            filename: 'index-[hash].html',
            //生成的index.html文件
            template: 'index.html',
            //目标文件
            inject: 'head'// script 放在哪个标签中
        })
    ]
```

还可以在其中加入其他参数 title content
直接在html 获取

```
<!DOCTYPE html>
<html>
<head>
    <title><%= htmlWebpackPlugin.options.title %></title>
</head>
<body>
<%= htmlWebpackPlugin.options.title %>
<%= htmlWebpackPlugin.options.content %>
<%= htmlWebpackPlugin.options.date %>
<% for (var key in htmlWebpackPlugin.files) {%>
<%= key %> : <%= JSON.stringify(htmlWebpackPlugin.files[key]) %>
<% } %>
<% for (var key in htmlWebpackPlugin.options) {%>
<%= key %> : <%= JSON.stringify(htmlWebpackPlugin.options[key]) %>
<% } %>
```

```
</body>
</html>
```

8.压缩使用参数minify 如

```
                module.exports = {
    // 打包源文件也可以接收数组['a.js', 'b.js'] 既打包合并两个文件
    // 也可以是键值
    // entry: {
    //      main: 'a.js',
    //      b: 'b.js'
    // }
    entry: './src/script/main.js',
    //生成目标文件
    output: {
        // 生成目标文件目录
        path: './dist',
        // 目标文件文件名 如果entry是一个对象则使用[name].bundle.js
        filename: 'js/[name].js',
        // 上线时候写的
        publicPath: 'http://cdn.com/'
    },
    plugins: [
        new htmlWebpackPlugin({
            // 打包之后的文件名
            filename: 'index.html',
            template: 'index.html',
            inject: 'head',// script 放在哪个标签中
            title: 'welcome CJJ123456',
            date: new Date(),
            content: 'sssss',
            // 压缩
            minify: {
                removeComments: true
            }
        })
    ]

}
```

9.多个js文件，过个html文件怎样分别引入不同的js

```
var htmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
    // 打包源文件也可以接收数组['a.js', 'b.js'] 既打包合并两个文件
    // 也可以是键值
    // entry: {
```

```
        //      main: 'a.js',
        //      b: 'b.js'
        // }
        entry: {
                main: './src/script/main.js',
                a: './src/script/a.js',
                b: './src/script/b.js'
        },
        //生成目标文件
        output: {
                // 生成目标文件目录
                path: './dist',
                // 目标文件文件名 如果entry是一个对象则使用[name].bundle.js
                filename: 'js/[name].js',
                // 上线时候写的
                publicPath: 'http://cdn.com/'
        },
        plugins: [
                new htmlWebpackPlugin({
                        // 打包之后的文件名
                        filename: 'a-[hash].html',
                        template: 'index.html',
                        inject: 'head',// script 放在哪个标签中
                        title: 'welcome A',
                        chunks: ['a'],//html 所引入的js文件
                        // excludeChunks: ['a'] 排除哪个js不引入
                }),
                new htmlWebpackPlugin({
                        // 打包之后的文件名
                        filename: 'b-[hash].html',
                        template: 'index.html',
                        inject: 'head',// script 放在哪个标签中
                        title: 'welcome B',
                        chunks: ['b']
                })
        ]

}
```

10.如果不想使用src引入js文件，而是直接插入到html文件中。可使用如下方式 其他js以链接的形式引入

注意在webpack.config.js中inject:false

```
<!DOCTYPE html>
<html>
<head>
        <title><%= htmlWebpackPlugin.options.title %></title>
        <script type="text/javascript">
```

```
                <%=
compilation.assets[htmlWebpackPlugin.files.chunks.main.entry.substr(htmlWebpack
Plugin.files.publicPath.length)].source() %>
    </script>
</head>
<body>
<% for (var key in htmlWebpackPlugin.files.chunks) {%>
    <% if (key !== 'main') {%>
    <script src="<%= htmlWebpackPlugin.files.chunks[key].entry %>" type="text/
javascript" ></script>
    <% } %>
<% } %>

</body>
</html>
```

11.Loader的使用见官网

babel-loder 转换ES6语法成浏览器可以直接运行的Javascript

首先js得loader 见babel官网，点击安装设置，点击web pack

安装cnpm install --save-dev babel-loader babel-core

配置文件如下

```
var htmlWebpackPlugin = require('html-webpack-plugin');
var path = require('path');

module.exports = {
    // 打包源文件也可以接收数组['a.js', 'b.js'] 既打包合并两个文件
    // 也可以是键值
    // entry: {
    //     main: 'a.js',
    //     b: 'b.js'
    // }
    entry:'./src/app.js',
    //生成目标文件
    output: {
        // 生成目标文件目录
        path: './dist',
        // 目标文件文件名 如果entry是一个对象则使用[name].bundle.js
        filename: 'js/[name].bundle.js',
    },
    module: {
        loaders: [
            {
                test: /\.js$/,
                loader: 'babel-loader',
                exclude: path.resolve(__dirname, 'node_modules'), // 排除的文
件
```

```
                include: path.resolve(__dirname, 'src'),
                // include: './src/',// 打包的位置
                query: {
                    presets: ["es2015"]
                }
            }
        ]
    },
    plugins: [
        new htmlWebpackPlugin({
            // 打包之后的文件名
            filename: 'index.html',
            template: 'index.html',
            inject: 'body',
            title: 'welcome CJJ',
            chunks: ['main']//html 所引入的js文件
        })
    ]

}
```

12.css-loader 和 style-loader

cnpm install style-loader css-loader --save-dev

// 解决浏览器前缀

cnpm install postcss-loader --save-dev

cnpm install autoprefixer --save-dev

cnpm install precss --save-dev

loader的处理方式是从右向左

web pack 对于css文件中@import 引入怎处理

loader: 'style-loader!css-loader?importLoaders=1!postcss-loader'

注意Sublime text 3中使用less 除了安装less2css 之外还需安装less-plugin-clean-css 和 less-plugin-autoprefix less不然会报 less2css error: `lessc` is not available

cnpm install -g less-plugin-clean-css　　cnpm install -g less-plugin-autoprefix

cnpm install -g less

13.模板文件引入html-loader等等（ejs）

cnpm install html-loader —save-dev

14.图片文件的处理

在css文件中指定相对路径的图片 background: url('../../assets/bg.png');

cnpm install file-loader --save-dev

```
{
    test: /\.(png|jpg|gif|svg)$/,
    loader: 'file-loader'
}
```

如果想要改变打包位置则需加入query参数

```
{
    test: /\.(png|jpg|gif|svg)$/i,
```

```
        loader: 'file-loader',
        query: {
                name: 'assets/[name]-[hash:5].[ext]'
            }
}
```

图片是一个base64则需要使用url-loader (增加代码的冗余)

```
cnpm install url-loader --save-dev

{
        test: /\.(png|jpg|gif|svg)$/i
        loader: 'url-loader',
        query: {
                name: 'assets/[name]-[hash:5].[ext]'
            }
}
cnpm install image-webpack-loader --save-dev
{
                        test: /\.(png|jpg|gif|svg)$/i,
                        loaders: [
                                'url-loader?limit=10000&name=assets/[name]-[hash:5].
[ext]',
                        'image-webpack-loader'
                        ]
                        // loader: 'url-loader'
                }
```