# WEBPACK的配置

环境版本说明：

node ： v10.16.0

yarn： 1.16.0

webpack: 4.33.0

打包下载环境：

链接：https://pan.baidu.com/s/1iFKoJ8wHIT9FhwwW8DjOEA

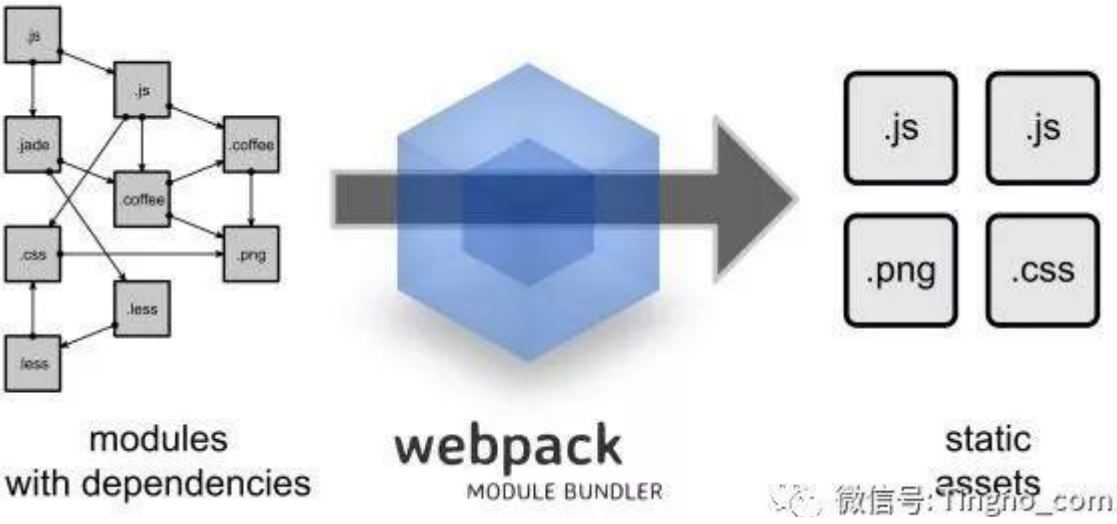
提取码：cztv

什么是WebPack?

WebPack可以看做是模块打包机：它做的事情是，分析你的项目结构，找到JavaScript模块以及其它的一些浏览器不能直接运行的拓展语言（Sass，TypeScript等），并将其转换和打包为合适的格式供浏览器使用。

Webpack的主要优势：

- 模块化开发（import，require）
- 预处理（Less，Sass，ES6，TypeScript,……）
- 压缩js代码
- 复制
- 自动处理CSS3属性前缀
- 单文件，多文件打包
- 热更新
- ……



一：安装

二：初始化

三：创建webpack.dev.js配置文件

四：配置入口文件

五：设置参数 通过script

六：CSS文件打包

七： 自动处理CSS3属性前缀

八 ： 给webpack增加babel支持

九：打包HTML文件

十： CSS中的图片处理

十一：JS压缩

准备工作：

1.安装node.js　安装完成之后，就自动生成npm(包管理器)

2. node -v (测试node的版本号)， npm -v(测试npm的版本号)

3.安装cnpm（类似于npm）

3.1 优势：它的服务器是在中国，运行速度会比较快

2.1 地址：https://npm.taobao.org/

2.2 在终端执行： sudo npm install -g cnpm --registry=https://registry.npm.taobao.org

或者安装yarn

## 一：安装

yarn global add webpack webpack-cli

yarn add webpack webpack-cli —dev

或者： npm install webpack webpack-cli -g

npm install webpack webpack-cli —save-dev

## 二：初始化

输入命令npm init -y,生成package.json

输入命令mkdir static src pages dev创建四个文件夹

输入命令touch dev/index.html src/index.js分别dev和src文件夹下面创建一个index.html文件和index.js文件

webpack4.x中打包默认找src/index.js作为默认入口，可以直接在终端中输入命令webpack 将当前的内容进行一个简单的打包，这时候看下你的项目目录dev文件下是不是多了一个main.js文件

输入命令rm dev/main.js src/index.js,再执行是会报错的，因为webpack默认的index.js被删除了

mode是webpack中独有的，有两种打包环境，一个是开发环境：development另外一个是生产环境：production

打包的时候输入

webpack —mode=development 或者

webpack —mode=production (自动压缩了)就不会出现警告提示了

## 三：创建webpack.dev.js配置文件

输入命令touch webpack.dev.js创建文件

```
module.exports = {
mode:’development’,
resolve:{
      extensions:['.less','.js','jsx']
  },

entry:{
//入口文件配置
main:path.resolve(__dirname,’src/main.js')
```

```
},
output:{
//出口文件配置
path:path.resolve(__dirname,”build”),
filename:’test.js’
},
//模块，例如解读css,图片如何转换，压缩等
modules:{},
//插件，用于生产模板和各项功能
plugins:{},
//配置webpack开发服务
devServer:{}
}
```

**配置**:

```
"scripts": {
    "build": "webpack --mode production --config webpack.config.js"
  }
```

执行npm run build

## 四. 配置入口文件

```
//入口文件的配置项
```

```
    entry:{
        //里面的main是可以随便写的
        main:'./src/main.js',
        main2:'./src/main2.js' //这里新添加一个入口文件
    },
```

多个入口文件：

```
var glob=require('glob');
var newEntries={};
/**
 * 动态查找所有入口文件
 */
var files = glob.sync(path.join(srcPath, 'js/*.jsx'));
files.forEach(function(file){
    var substr = file.match(/src\/js\/(\S*)\.jsx/)[1];
    newEntries[substr] = file;
});
```

我们会发现报错提示 Conflict: Multiple assets emit to the same filename bundle.js 翻译过来告诉我们 冲突:多个资产发出相同的文件名bundle.js。这个时候就需要我们来配置出口文件了，下面是之前的出口文件

现在我们将原来写死的test.js给改成了 [name].js

注：[name]的意思是根据入口文件的名称，打包成相同的名称，有几个入口文件，就可以打包出几个文件。

## 五. 设置参数 通过script

yarn add cross-env —dev

```
"scripts": {
    "dev": "cross-env scene=dev webpack-dev-server --config webpack.config.js",
    "build": "cross-env scene=prod webpack --mode production --config webpack.config.js"
  },
```

## 六：设置webpack-dev-server

yarn add webpack-dev-server —dev

```
webpackdevServer={
        //目录位置，用于找到程序打包地址
        contentBase: buildPath,
        historyApiFallback: true,
       //服务器的IP地址，可以使用IP也可以使用localhost
        host: '0.0.0.0',
```

```
    //服务端口
     port: 3008
  };
```

npm run dev 可以跑起来，然后我们再到项目的dev文件夹下面的index.html 文件中引入打包的两个js, 打开：localhost:3008/index.html

## 六. CSS文件打包

打包css需要下载配置css 的loader：style-loader 和 css-loader

在终端输入命令 yarn add style-loader css-loader —dev

### webpack.dev.config.js中对module属性中的代码进行配置

```
{
            test: /\.css$/,
            use: [{
                loader: "style-loader"},
               {
                 loader: "css-loader" }
           ]
},
```

或者：

```
{
            test: /\.css$/,
```

```
        use: [‘style-loader’，’css-loader’ ]
},
```

或者：

```
{
        test: /\.css$/,
        loader:'style-loader!css-loader',
},
```

**Less文件的打包和分离**

Less 是一门 CSS 预处理语言，它扩展了 CSS 语言，增加了变量、Mixin、函数等特性，使 CSS 更易维护和扩展

yarn less less-loader —dev

```
{
    test: /\.less$/,
    use: ['style-loader','css-loader','less-loader']
   // use: [{  loader: "style-loader"    },
        //    { loader: "css-loader" },
        //    {  loader:  “less-loader" }]
      },
```

**SASS文件的打包和分离**

其实sass跟less 的配置很像，这里sass首先要安装两个包，node-sass和sass-loader,因为sass-loader依赖于node-sass，所以需要先安装node-sass

yarn add node-sass sass-loader

```
{
        test: /\.scss$/,
        use: [{  loader: "style-loader" },
          {loader: "css-loader"},
         {loader: "sass-loader"}]
      }
```

**Less scss分离**

yarn add extract-text-webpack-plugin —dev
extract-text-webpack-plugin@next

或者：mini-css-extract-plugin

```
const extractTextPlugin = require('extract-text-webpack-plugin');
//必须是最新的beta版本才支持webpack4

pluginsAll.push(new extractTextPlugin(‘styles/'+outFilename+'.css'));

{
        test: /\.scss$/,
        use: extractTextPlugin.extract({
```

```
            use:[{ loader: "css-loader" },
              { loader:"postcss-loader"},
              { loader: "sass-loader"}],
              fallback: "style-loader"
          })
        },
```

## 七。 自动处理CSS3属性前缀

首先需要安装两个包postcss-loader 和autoprefixer（自动添加前缀的插件）

yarn add postcss-loader autoprefixer

postCSS推荐在项目根目录（和webpack.config.js同级），建立一个postcss.config.js文件。【注意⚠️：一定呀建在根目录下，不然会报错】

```
module.exports={
  plugins: [
    require('autoprefixer') //自动添加前缀插件
  ]
}
```

在webpack.dev.cnfig.js中配置

```
{
    test:/\.css$/,
    use: extractTextPlugin.extract({
```

```
        fallback: "style-loader",
        use:[{loader:"css-loader"},
                {
                    loader:"postcss-loader",
                },
                ]
            })
        },
```

## 八 。给webpack增加babel支持

yarn add  babel-core babel-loader babel-preset-es2015

在webpack.dev.config.js中配置Babel的方法如下：

```
//babel 配置
        {
            test:/\.(jsx|js)$/,
            use:{
                loader:'babel-loader',
                options:{
                    presets:[
                        "es2015","react"
                    ]
                }
            },
```

```
            exclude:/node_modules/
        }
```

.babelrc配置

虽然Babel可以直接在webpack.config.js中进行配置，但是考虑到babel具有非常多的配置选项，如果卸载webapck.config.js中会非常的雍长不可阅读，所以我们经常把配置卸载.babelrc文件里。

在项目根目录新建.babelrc文件，并把配置写到文件里。

.babelrc文件下

```
{
    "presets":["es2015"]
}
```

这时候.webpack.dev.config.js里的loader配置

```
//babel 配置
        {
            test:/\.(jsx|js)$/,
            use:{
                loader:'babel-loader',
            },
            exclude:/node_modules/
        }
```

## 十。 CSS中的图片处理

yarn  add url-loader

解释下：

file-loader：解决引用路径的问题，拿background样式用url引入背景图来说，我们都知道，webpack最终会将各个模块打包成一个文件，因此我们样式中的url路径是相对入口html页面的，而不是相对于原始css文件所在的路径的。这就会导致图片引入失败。这个问题是用file-loader解决的，file-loader可以解析项目中的url引入（不仅限于css），根据我们的配置，将图片拷贝到相应的路径，再根据我们的配置，修改打包后文件引用路径，使之指向正确的文件。

url-loader：如果图片较多，会发很多http请求，会降低页面性能。这个问题可以通过url-loader解决。url-loader会将引入的图片编码，生成dataURl。相当于把图片数据翻译成一串字符。再把这串字符打包到文件中，最终只需要引入这个文件就能访问图片了。当然，如果图片较大，编码会消耗性能。因此url-loader提供了一个limit参数，小于limit字节的文件会被转为DataURl，大于limit的还会使用file-loader进行copy。

```
//图片 loader
        {
            test:/\.(png|jpg|gif|jpeg)/, //是匹配图片文件后缀名称
            use:[{
                loader:'url-loader', //是指定使用的loader和loader的配置参数
                options:{
                    limit:500  //是把小于500B的文件打成Base64的格式，写入JS
```

```
            }
        }]
    }
```

注意：为什么只使用了url-loader

有的小伙伴会发现我们并没有在webpack.dev.config.js中使用file-loader，但是依然打包成功了。我们需要了解file-loader和url-loader的关系。url-loader和file-loader是什么关系呢？简答地说，url-loader封装了file-loader。url-loader不依赖于file-loader，即使用url-loader时，只需要安装url-loader即可，不需要安装file-loader，因为url-loader内置了file-loader。通过上面的介绍，我们可以看到，url-loader工作分两种情况：

1.文件大小小于limit参数，url-loader将会把文件转为DataURL（Base64格式）；

2.文件大小大于limit，url-loader会调用file-loader进行处理，参数也会直接传给file-loader。

也就是说，其实我们只安装一个url-loader就可以了。但是为了以后的操作方便，我们这里就顺便安装上file-loader。

## 十一. JS压缩

yarn add uglifyjs-webpack-plugin 或者

yarn add webpack-parallel-uglify-plugin。//压缩js，提高压缩速度

方式一：

```
pluginsAll.push(new UglifyJsPlugin({
        uglifyOptions: {
            compress: {
            drop_console: true,
            drop_debugger: true
            }
        }
    }));
```

方式二：

```
var ParallelUglifyPlugin = require('webpack-parallel-uglify-plugin');

pluginsAll.push(new ParallelUglifyPlugin({
        cacheDir: packageRoute.outputPath+'/.cache/',
        uglifyJS:{
            output: {
                comments: false
            },
            compress: {
                warnings: false,
                drop_console: true,
```

```
            drop_debugger: true
        }
    }
}));
```

## 九。打包HTML文件

yarn add  html-webpack-plugin

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

new htmlWebpackPlugin({
        minify:{
            //是对html文件进行压缩
            removeAttributeQuotes:true
           //removeAttrubuteQuotes是却掉属性的双引号。
        },
        hash:true,
         //为了开发中js有缓存效果，所以加入hash，这样可以有效避免缓存JS。
        template:'./src/index.html'
       //是要打包的html模版路径和文件名称。
    })
```

npm install --save-dev extract-text-webpack-plugin@next

npm i webpack webpack-cli -D

下载指定webpack指定版本：

npm i -D webpack@3

npm i -D 是 npm install --save-dev 的简写，是指安装模块并保存到

二.webpack配置文件常用配置项介绍

4.entry详细说明

（1）当entry是一个字符串时，这个字符串表示需要打包的模块的路径,如果只有一个要打包的模块，可以使用这种形式

（2）当entry是一个对象

a.是数组时,当需要将多个模块打包成一个模块，可以使用这个方式。如果这些模块之间不存在依赖，数组中值的顺序没有要求，如果存在依赖，则要将依赖性最高的模块放在最后面。

例如：entry:["./app/one.js",".app/two.js"]

b.是键值对形式的对象是，当需要分别打包成多个模块时，可以使用这种方式，例

如；

```
  entry:{
        module1:"./app/one.js",
        module2:["./app/two.js","./app/three.js"]
}
```

**dependencies和devDependencies的区别?**

对于我们依赖的这些插件库，有的是我们开发所使用的，有的则是项目所依赖的。对于这个分界线，我们诞生了dependencies和devDependencies，具体区别如下：

devDependencies：开发环境使用
dependencies：生产环境使用
例如：webpack，gulp等打包工具，这些都是我们开发阶段使用的，代码提交线上时，不需要这些工具，所以我们将它放入devDependencies即可，但是像jquery这类插件库，是我们生产环境所使用的，所以如要放入dependencies，如果未将jquery安装到dependencies，那么项目就可能报错，无法运行，所以类似这种项目必须依赖的插件库，我们则必须打入dependencies中

## 打包javascript模块

支持的js模块化方案包括：

ES6 模块

```
import MyModule from './MyModule.js';
```

CommonJS

```
var MyModule = require('./MyModule.js');
```

AMD

```
define(['./MyModule.js'], function (MyModule) {});
```

## 公用的模块分开打包

这需要通过插件“CommonsChunkPlugin”来实现。这个插件不需要安装，因为webpack已经把他包含进去了。接着我们来看配置文件：

```
var config = {
    entry:{app:path.resolve(__dirname,'src/main.js'),
            vendor: ["./src/js/common"]},//【1】注意这里
    resolve:{
        extentions:["","js"]
    },
    output:{
        path:buildPath,
        filename:"app.js"
    },
```

```
    plugins:[
        //【2】注意这里  这两个地方市用来配置common.js模块单独打包的
        new webpack.optimize.CommonsChunkPlugin({
            name: "vendor",//和上面配置的入口对应
            filename: "vendor.js"//导出的文件的名称
        })
    ]
}
```