**webpack 使用**

webpack结合gulp使用

1. 安装gulp-webpack-stream、webpack（在webpack.config.js中用到）
2. gulp中需要配置webpack的输入路径，输出路径，以及webpack的配置流。代码如下：

var del = require('del');

var webpack = require('webpack-stream');

// vinyl-named 可以在gulp中对生成文件命名，或者通过webpack.config中的output来命名

var named = require('vinyl-named');

// var path = require('path');

// var fs = require('vinyl-fs');

var vinylPaths = require('vinyl-paths');

var webpackReg = /src(.\*)\/.\*\.(\w+)/;;

gulp.task('webpack', (cb) => {

// console.log(path.join(\_\_dirname, 'src/testPath.js'));

// webpack-stream 无法根据输入路径，返回dest路径，只会输出到dest

del('dist/\*', cb);

// 根据输入路径，输出到不同目录

gulp.src(['src/js/\*\*/\*.js',

'src/js/\*\*/\*.es6',

'src/css/\*\*/\*.less',

'!src/build/\*.js'])

.pipe(vinylPaths(function (paths) {

var match = paths.replace(/\\/g, '/').match( webpackReg );

// 文件是一个个传入，故common插件无法合并相同require代码

if(match.length){

gulp.src(match[0])

.pipe(named())

.pipe(webpack(require('./webpack.config.js')))

.pipe(gulp.dest('dist/'+match[1]));

}

return Promise.resolve();

}))

// common插件有效

// gulp.src(['src/js/testExports\_B.js', 'src/js/testRequire\_A.js'])

// .pipe(named())

// .pipe(webpack(require('./webpack.config.js')))

// .pipe(gulp.dest('dist/'));

});

**关于输入路径和输出路径的说明：**

在gulp和webpack结合使用的时候，有两种配置路径的方式。

1. 通过webpack.config.js来配置。在config文件中添加entry和output。这种方式，最后生成的文件会以entry中的key替换output中的[name]来命名。如果需要将输出文件放入不同的文件夹，则需要在key的命名时添加输出路径，

如：js/A1: 'js/testRequire\_A'

缺点：如果需要配置大量入口文件，则会相当麻烦。适用于single-page的项目中使用。不能使用\*\*通配符

entry: {

A1: 'js/testRequire\_A',

B1: "js/testExports\_B"

},

output: {

filename: "dist/[name].chunk.js"

},

1. 通过gulp中的gulp.src().pipe( named() ).pipe( gulp.dest() ) 的方式来配置。

当中named() 是为保证输出文件的文件名与输入文件保持一致。用这种方式，webpack.config.js文件中不需要配置entry和output。同时在入口文件中可以使用\*通配符。但是还有一个问题，就是打包输出的文件，只能在同一个文件夹中，无法做到与输入路径一致。（其他插件，如js压缩，输出文件的路径，会根据输入文件的通配符路径，与输入文件保持一致）

目前的解决方法为，通过vinyl-paths插件，获取每个输入文件的路径，然后每个文件分别做webpack处理输出。但又会出现一个新的问题，因为是分别单独输入输出的，就无法使用webpack的new webpack.optimize.CommonsChunkPlugin("common.js")插件。会在每个输出目录中都产生一个common.js文件，又在每个文件编译时相互覆盖。最终没有任何效果，还没有有效的解决方案。（CommonsChunkPlugin用于将所有入口文件中相同的require放到common.js文件中，然后在html中单独引入common.js以减少输出文件的大小）

**最终&最优，也是最正统的方案（2016-8-30）**

文件通过named方法，返回带有路径的文件名。文件一次性打包，解决了图片加载时，不同层级下图片路径不一致的问题。同时common.js也可以正常使用。

var path = require('path');

var webpackReg = /src\/(.\*)\.(\w+)/;;

gulp.src(['src/js/\*\*/\*.js',

'src/js/\*\*/\*.es6',

'src/css/\*\*/\*.less',

'src/css/\*\*/\*.scss',

'!src/build/\*.js'])

.pipe(named(function(file){

var match = file.path.replace(/\\/g, '/').match( webpackReg );

var fileName = path.basename(file.path, path.extname(file.path));

if(match.length){

fileName = match[1] ;

}

return fileName;

}))

.pipe(webpack(require('./webpack.config.js')))

.pipe(gulp.dest('dist/'));

1. webpack.config.js文件的配置

var path = require('path');

var webpack = require('webpack');

module.exports = {

watch: true, // 侦听文件变化自动编译

devtool: "source-map", // 使用source-map，定位原文件

resolve: {

// 定义在require时，不需要输入的扩展名

extensions: ['', '.js', '.jsx', '.es6', '.less', '.css'],

// 获取项目的绝对路径，用于设置项目根目录

root: path.join(\_\_dirname, 'src/'),

// 设置文件别名，可直接通过key值require到。

// 别名中无法使用$等特殊符号

alias: {

testExports: path.join(\_\_dirname, 'src/js/testExports\_B'),

jquery: path.join(\_\_dirname, 'src/build/jquery-1.11.2')

}

},

// 用于加载外部js文件，不会打包到最终生成文件中。第三方库建议使用这种方式

// 需要在 html 文件中添加<script jquery>

externals: {

"$": "jQuery"

},

plugins: [

// 会将入口文件中相同的require内容打包到common.js中

// new webpack.optimize.CommonsChunkPlugin("common.js")

],

// 在js文件中可直接使用 $ 不需要再require('jquery')引入。

// 但会打包到最终生成文件中，增加每个文件的大小。对第三方库不建议使用

plugins: [

new webpack.ProvidePlugin({

"$": "jquery"

})

],

// 资源加载，通过安装xxx-loader插件来加载。

module: {

loaders: [{

test: /\.jsx$/, // 验证输入文件后缀

loader: 'babel', // 使用的加载器，可不加-loader，多个用 ！连接

query: { // 加载器使用需要的预置项或插件

presets: ['es2015', 'react'],

plugins: ['transform-runtime']

}

}]

}

};

在js中引入模块的时候，如require(‘css/index.less’)，不能使用变量

（如：var index = ‘index’; require(‘css/’+index+’.less’) 不能正确识别路径）

**context**

webpack处理entry选项时的基础路径（绝对路径），默认值为process.cmd()与\_\_dirname相同，即webpack.config.js文件所在绝对路径

**es6编译：**

安装babel-loader、babel-preset-es2015、babel-preset-react、babel-plugin-transform-runtime（可动态添加使用到的es6语法，在js头部require即可） 或 babel-preset-stage-0（用于解析class中的属性语法）

**less编译加载：**

安装：less、less-loader、css-loader、style-loader

使用：loader: "style!css!less"

**sass编译加载：**

安装：Webpack的sass-loader还是依赖于node-sass以及sass(gem)，所以如果安装sass-loader报错可以先尝试安装sass。需要的安装项如下：

ruby环境，gem install sass安装sass， 以及sass-loader、node-sass、css-loader、style-loader

**图片加载：**

安装：file-loader、url-loader

使用：loader: 'url-loader?limit=10000&name=/dist/images/[name].[ext]' 如果图片资源小于10kb就会转化成 base64 格式的 dataUrl，其他的图片会存放在 /dist/images/ 目录下。后面引用资源的时候都会使用这个路径，因为源文件是相对于js或css的路径，而生成的文件是相对于html的，所以使用绝对路径相对稳妥。

JS中图片通过require加载的，也可以编译到 /dist/images/ 目录下，并在生成文件中引用新的路径。

**补充：**loader: 'url-loader?limit=10000&name=images/[name].[ext]'（2016-8-30）

因打包方式不同，上述介绍有误。使用绝对路径的话，实际打包图片目录会置于网站目录外层。而使用相对目录，则可在gulp.dest()的输出目录中生成图片文件夹images。然后配合output: { publicPath: '/dist/' } 使用。编译后生成的文件，调用的图片路径为 publicPath+name，

即 '/dist/ images/[name].[ext]'

可参考：

<http://www.tuicool.com/articles/UzIJFvB>

<http://www.07net01.com/2015/08/890558.html>

<http://webpack.github.io/docs/configuration.html>

**附 gulp中编译sass：**

gulp-ruby-sass是调用sass，所以需要ruby环境，需要生成临时目录和临时文件。需要先下载并安装Ruby的windows安装包，添加环境变量。

然后gem install sass安装sass。npm install gulp-ruby-sas gulp-sourcemaps。使用方法：

var sass = require('gulp-ruby-sass'),

sourcemaps = require('gulp-sourcemaps');

gulp.task('sass', function() {

return sass('./src/sass/\*\*/\*.scss', {

sourcemap: true,

// 嵌套输出方式 nested, 展开输出方式 expanded, 紧凑输出方式 compact, 压缩输出方式 compressed

style: 'expanded'

})

.on('error', sass.logError)

// 在css文件中写souremaps

// .pipe(sourcemaps.write())

// 在.map文件中写sourcemaps

.pipe(sourcemaps.write('./', { // map文件的路径, 相对于css文件

includeContent: false,

sourceRoot: '../sass' // scss文件的路径, 相对于map文件

}))

.pipe(gulp.dest('./src/css'));

});

gulp-sass是调用node-sass,有node.js环境就够了，编译过程不需要临时目录和文件，直接通过buffer内容转换。