# 安装

npm install webpack –g 全局安装

npm install webpack --save-dev 项目安装

# 配置webpack.config.js

详细配置：

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

例子：

var webpack = require('webpack');

var commonsPlugin = new webpack.optimize.CommonsChunkPlugin('common.js');

module.exports = {

//插件项

plugins: [commonsPlugin],

//页面入口文件配置

entry: {

index : './src/js/page/index.js'

},

//入口文件输出配置

output: {

path: 'dist/js/page',

filename: '[name].js'

},

module: {

//加载器配置

loaders: [

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

{ test: /\.js$/, loader: 'jsx-loader?harmony' },

{ test: /\.scss$/, loader: 'style!css!sass?sourceMap'},

{ test: /\.(png|jpg)$/, loader: 'url-loader?limit=8192'}

]

},

//其它解决方案配置

resolve: {

root: 'E:/github/flux-example/src', //绝对路径

extensions: ['', '.js', '.json', '.scss'],

alias: {

AppStore : 'js/stores/AppStores.js',

ActionType : 'js/actions/ActionType.js',

AppAction : 'js/actions/AppAction.js'

}

}

};

## plugins是插件项

数组引入

## entry 是页面入口文件配置

如：

{

entry: {

page1: "./page1",

//支持数组形式，将加载数组中的所有模块，但以最后一个模块作为输出

page2: ["./entry1", "./entry2"]

},

output: {

path: "dist/js/page",

filename: "[name].bundle.js"

}

}

该段代码最终会生成一个 page1.bundle.js 和 page2.bundle.js，并存放到 ./dist/js/page 文件夹下。

## output 输出项配置

即入口文件最终要生成什么名字的文件、存放到哪里

* **path**: 打包文件存放的绝对路径包括图片
* **publicPath**: 网站运行时的访问路径
* **filename**: 打包后的文件名 通常是 [name].bundle.js

## module.loaders

告知 webpack 每一种文件都需要使用什么加载器来处理。

如：

module: {

//加载器配置

loaders: [

//.css 文件使用 style-loader 和 css-loader 来处理

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

//.js 文件使用 jsx-loader 来编译处理

{ test: /\.js$/, loader: 'jsx-loader?harmony' },

//.scss 文件使用 style-loader、css-loader 和 sass-loader 来编译处理

{ test: /\.scss$/, loader: 'style!css!sass?sourceMap'},

//图片文件使用 url-loader 来处理，小于8kb的直接转为base64

{ test: /\.(png|jpg)$/, loader: 'url-loader?limit=8192'}

]

}

如上，"-loader"其实是可以省略不写的，多个loader之间用“!”连接起来，

所有的加载器都需要通过 npm 来加载

## resolve

resolve: {

//查找module的话从这里开始查找

root: 'E:/github/flux-example/src', //绝对路径

//自动扩展文件后缀名，意味着我们require模块可以省略不写后缀名

extensions: ['', '.js', '.json', '.scss'],

//模块别名定义，方便后续直接引用别名，无须多写长长的地址

alias: {

AppStore : 'js/stores/AppStores.js',//后续直接 require('AppStore') 即可

ActionType : 'js/actions/ActionType.js',

AppAction : 'js/actions/AppAction.js'

}

}

# 运行

$ webpack --display-error-details

即可，后面的参数“--display-error-details”是推荐加上的，方便出错时能查阅更详尽的信息（比如 webpack 寻找模块的过程），从而更好定位到问题。

## 参数

webpack --config XXX.js //使用另一份配置文件（比如webpack.config2.js）来打包

webpack --watch //监听变动并自动打包

webpack -p //压缩混淆脚本，这个非常非常重要！

webpack -d //生成map映射文件，告知哪些模块被最终打包到哪里了

# 加载文件和插件

## CommonsChunkPlugin

提取多个入口文件的公共脚本部分，然后生成一个 common.js 来方便多页面之间进行复用

## 加载css

需要style-loader和csss-loader

### 安装

npm install style-loader csss-loader -save-dev

### 配置

module: {

loaders:[

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

]

}

在文件中引入：（如main.css）  
require(‘main.css’)

### 模块css

配置：

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

require(‘.css’) 才能返回对象

var style = require('../css/main.css');

ReactDOM.render(

<div>

<h1 className={style.red}>局部</h1> //这里运用

<span className="blue ">全局</span>

</div>,

document.getElementById('box')

)

**main.css:**

.red {

color: red

}

:global(.blue){

color: blue;

}

## 加载图片

需要url-loader

将样式中引用到的图片转为模块来处理

### 安装

npm install url-loader -save-dev

### 配置

module: {

loaders:[

{ test: /\.(png|jpg)$/, loader: 'url-loader?limit=8192&name=images/[name].[ext] ' }

]

}

配置信息的参数“?limit=8192”表示将所有小于8kb的图片都转为base64形式

name=images/[name].[ext] 大于limit的图片会输出的路径

在css中引入如img标签

<img src=”images/xx.xx” />

## 加载react

需要jsx，jsx-loader，react，react-dom

### 安装

npm install jsx jsx-loader react react-dom –save -dev

### 配置

module: {

loaders:[

{ test: /\.js[x]?$/, loaders: ['jsx?harmony'] }

]

}

### 使用

require(‘react’);

require(‘react-dom’);

## 压缩js

需要webpack中的UglifyJsPlugin

### 安装

npm install webpack –save –dev 即可

### 使用

放在plugins里面。

var webpack = require('webpack');

var uglifyJsPlugin = webpack.optimize.UglifyJsPlugin;

plugins:[

new uglifyJsPlugin({

compress: {

warnings: false

}

})

],

## 自动生成index.html和自动打开浏览器

需要

html Webpack Plugin 生成html（不需要自己写index.html）

和

Open Browser Webpack 自动打开浏览器

### 安装

npm install html-webpack-plugin open-browser-webpack-plugin –save

### 配置

webpack.config.js

var HtmlwebpackPlugin = require('html-webpack-plugin');

var OpenBrowserPlugin = require('open-browser-webpack-plugin');

module.exports = {

plugins:[

new HtmlwebpackPlugin({

title: 'Webpack-demos',

filename: 'index.html'

}),

new OpenBrowserPlugin({

url: 'http://localhost:8080'

})

],

}

## 运行环境设置

需要webpack

### 安装

npm install webpack –save –dev

### 配置

webpack.config.js：

var devFlagPlugin = new webpack.DefinePlugin({

\_\_DEV\_\_: JSON.stringify(JSON.parse(process.env.DEBUG || 'false'))

});

module.exports = {

plugins: [devFlagPlugin]

};

**在main.js中**

用if(\_\_DEV\_\_) 判断 ，上面配置的参数

**启动需要改一下**

# Linux & Mac

$ env DEBUG=true webpack-dev-server

# Windows

$ set DEBUG=true

$ webpack-dev-server

## 分离代码

### require.ensure

require.ensure(['./a'], function(require) {

var content = require('./a');

document.open();

document.write('<h1>' + content + '</h1>');

document.close();

});

require.ensure 定义一个分离点，告诉webpack ./a.js从bundle.js中分离到另一个文件中

### bundle-loader

var load = require('bundle-loader!./a.js');

load(function(file) {

document.open();

document.write('<h1>' + file + '</h1>');

document.close();

});

## 提取公共部分

使用插件CommonsChunkPlugin

### 配置

var CommonsChunkPlugin = require("webpack/lib/optimize/CommonsChunkPlugin");

module.exports = {

plugins: [

new CommonsChunkPlugin('init.js')

]

}

webpack会把js相同部分合并到init.js中