**官方文档: <http://webpack.github.io/docs/configuration.html>**

# **webpack基础配置**

**http://www.cnblogs.com/bldxh/p/6104192.html**

1. webpack运行规则

Webpack 会给每个模块分配一个唯一的id并通过这个id索引和访问模块。在页面启动时，会先执行入口文件中的代码，其它模块会在运行 require 的时候再执行。

1. 运行时主要的参数

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

webpack --watch //监听变动并自动打包

webpack -p //压缩混淆脚本

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

webpack --display-error-details //打印错误详情。

还有一些看性能的参数：

webpack --colors //输出结果带彩色，比如：会用红色显示耗时较长的步骤

webpack --profile //输出性能数据，可以看到每一步的耗时

webpack --display-modules //默认情况下 node\_modules 下的模块会被隐藏，加上这个参数可以显示这些被隐藏的模块

webpack entry.js bundle.js //编译 entry.js 并打包到 bundle.js

1. 配置文件

整个配置文件就是一个大的json，这个json需要用module.exports导出, 常用有如下配置:

* entry:{}
* output:{}
* module:{}
* resolve:{}
* plugins:[]

1. entry

值可以为String,[String],Object

{

entry1: "./page1",

entry2: ["./page2", "./page3"]

}

value为数组形式时，将加载数组中的所有模块，但**以最后一个模块作为输出**。

1. output

{

path: './build', //决定文件在磁盘中的位置,最好是绝对路径

filename: 'bundle.js'// 每个文件自己唯一的名字

//最后打包出来的文件位置: ./build/bundle.js

publicPath: 'http://mycdn.com/', // 用来生成URL

chunkFilename:'',//非入口块的文件名

sourceMapFilename:'',

//SourceMaps的文件名,默认值是 `[file].map`

library:'',//单独打包自己写的某个库，值为引用时需要的模块名

libraryTarget:'',//规定库输出的格式，这两个参数一般在生成工具的配置文件中，不在项目打包文件中

}

filename可用变量：[name],[hash],[chunkhash]

chunkFilename可用变量：[id],[name],[hash],[chunkhash]

sourceMapFilename可用变量：[file],[id],[hash]

libraryTarget可选值：

'var'(default),'this','commentjs','commentjs2','amd','umd'

1. module

loaders:[]

{

test: 需要匹配的文件类型，用正则,

loader: 需要使用的加载器,

enclude:''/[],

include:''/[], l

oader: 用!分割加载器,

loaders: 一个加载器的数组

},

(1)和loaders相似的两个参数：

module.preLoaders, module.postLoaders，区别只在于时机不同

(2)loader和loaders

①加载器需要单独下载，'-loader'可以省略不写,这个命名规则和搜索优先级顺序在 webpack 的 resolveLoader.moduleTemplates api 中定义。

Default: ["-webpack-loader", "-web-loader", "-loader", ""]

②多个加载器处理同一类型文件用'!'连接

③loader的配置信息的参数写到'?'后面

④loader添加方法：

* 在 webpack 全局配置中进行绑定，写在配置文件中
* 引用模块的时候添加，如：require("!style!css!./style.css")
* 通过命令行的方式使用。比如webpack entry.js bundle.js --module-bind 'css=style!css'

Loader 可以同步或异步执行。

1. plugins

包括系统插件和扩展插件

CommonsChunkPlugin

作用：用于提取多个入口文件的公共脚本部分

entry: {

p1: "./page1",

p2: "./page2",

p3: "./page3",

ap1: "./admin/page1",

ap2: "./admin/page2"

},

output: {

filename: "[name].js"

},

plugins: [

new CommonsChunkPlugin("admin-commons.js", ["ap1", "ap2"]),

new CommonsChunkPlugin("commons.js", ["p1", "p2", "admin-commons.js"])

]

script--- required:

page1.html: commons.js, p1.js

page2.html: commons.js, p2.js

page3.html: p3.js

admin-page1.html: commons.js, admin-commons.js, ap1.js

admin-page2.html: commons.js, admin-commons.js, ap2.js

1. externals

externals: {

"react": "React",

} //对应于const React=require('react');

1. resolve

Webpack 的配置提供了 resolve 和 resolveLoader 参数来设置模块解析的处理细节，resolve 用来配置应用层的模块（要被打包的模块）解析，resolveLoader 用来配置 loader 模块的解析。

{ //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,注意一下, 第一个是空字符串! 对应不需要后缀的情况.

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,也可以省下不少搜索硬盘的时间。

alias: {

AppAction : 'js/actions/AppAction.js'

}

//后续直接require('AppAction') 即可

}

1. **高级使用：shimming**

<https://github.com/webpack/docs/wiki/shimming-modules>

在 AMD/CMD 中，我们需要对不符合规范的模块（比如一些直接返回全局变量的插件）进行 shim 处理，这时候我们需要使用 exports-loader 。

{

test:

require.resolve("./src/js/tool/swipe.js"),

loader: "exports?swipe"

}

之后在脚本中需要引用该模块的时候，这么简单地来使用就可以了：

require('./tool/swipe.js');  
 swipe();

使用 script.js 在脚本中来加载我们的模块：

var $script = require("scriptjs");

$script("//ajax.googleapis.com/ajax/libs/jquery/2.0.0/jquery.min.js", function() {

$('body').html('It works!')

});

# 入门Webpack，看这篇就够了

<http://www.jianshu.com/p/42e11515c10f#>

F:\mygit\webpack\webpackSample

1. 代码git地址

https://github.com/nmw1990/webpack.git

1. 本项目命令
2. webpack 打包开发环境(注释.babelic中的"env":{}键值对)
3. npm start打包本地服务器环境(热加载模块)
4. npm run build打包生产环境(优化，压缩，缓存等等)
5. 注意webpack2与webpack1的版本不同，对webpack.config.js中的设置不同，所对应安装的插件的版本也不同，本教程使用的webpack版本为1.12.9。

"webpack": "^1.12.9",

"webpack-dev-server": "^1.15.0"

#plugin for webpack 2

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

#plugin for webpack 1

npm install --save-dev extract-text-webpack-plugin@1.0.1

1. 安装指定版本的node\_modules包

方法一: npm install webpack@1.12.9 --save-dev

方法二:①修改package.json

"devDependencies": {"webpack": "^1.12.9"}

②npm install

1. Webpack2 升级指南和特性摘要

<https://segmentfault.com/a/1190000008181955>

1. 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（Scss，TypeScript等），并将其打包为合适的格式以供浏览器使用。

1. WebPack和Grunt以及Gulp对比

Grunt和Gulp的工作方式是：在一个配置文件中，指明对某些文件进行类似编译，组合，压缩等任务的具体步骤，这个工具之后可以自动替你完成这些任务。Gulp/Grunt是一种能够优化前端的开发流程的工具，而WebPack是一种模块化的解决方案。

1. Webpack的工作方式

把你的项目当做一个整体，通过一个给定的主文件（如：index.js），Webpack将从这个文件开始找到你的项目的所有依赖文件，使用loaders处理它们，最后打包为一个浏览器可识别的JavaScript文件。

1. \_\_dirname

“\_\_dirname”是node.js中的一个全局变量，它指向当前执行脚本所在的目录。

1. 简化命令

在命令行中使用npm start就可以执行相关命令，如果对应的此脚本名称不是start，想要在命令行中运行时，需要这样用npm run {script name}如npm run build，以下是执行npm start后命令行的输出显示。

scripts": {

"start": "webpack"

//相当于把npm start命令指向webpack命令

},

1. Source Maps

开发总是离不开调试，如果可以更加方便的调试当然就能提高开发效率，不过打包后的文件有时候你是不容易找到出错了的地方对应的源代码的位置的，Source Maps解决了这个问题的。(只在开发阶段使用它)

1. webpack高版本中-loader不允许简写

It's no longer allowed to omit the '-loader' suffix when using loaders.You need to specify 'json-loader' instead of 'json'.

1. --save-dev与--save的对比

npm install --save-dev 包名 与npm install --save 包名的区别

前者安装包，并写入webpack.json的devDependencies对象中；

后者安装包，并写入webpack.json的dependencies对象中。

1. webpack构建本地服务器

Webpack提供一个可选的本地开发服务器，这个本地服务器基于node.js构建，可以实现你想要的这些功能，不过它是一个单独的组件，在webpack中进行配置之前需要单独安装它作为项目依赖。

npm install --save-dev webpack-dev-server

1. Loaders

Loaders是webpack中最让人激动人心的功能之一。使用不同的loader，webpack通过调用外部的脚本或工具可以对各种各样的格式的文件进行处理，比如说分析JSON文件并把它转换为JavaScript文件，或者说把下一代的JS文件（ES6，ES7)转换为现代浏览器可以识别的JS文件。或者说对React的开发而言，合适的Loaders可以把React的JSX文件转换为JS文件。

Loaders需要单独安装并且需要在webpack.config.js下的modules关键字下进行配置。

**注释: ①Loader是把其他各种格式的文件转化为JavaScript格式。②使用loader需要先通过npm安装。**

1. CSS处理

webpack提供两个工具处理样式表，css-loader 和 style-loader，二者处理的任务不同，css-loader使你能够使用类似@import 和 url(...)的方法实现 require()的功能,style-loader将所有的计算后的样式加入页面中，二者组合在一起使你能够把样式表嵌入webpack打包后的JS文件中。

1. CSS modules

把JS的模块化思想带入CSS中来，通过CSS模块，所有的类名，动画名默认都只作用于当前模块。Webpack从一开始就对CSS模块化提供了支持，在CSS loader中进行配置后，你所需要做的一切就是把”modules“传递都所需要的地方，然后就可以直接把CSS的类名传递到组件的代码中，且这样做只对当前组件有效，不必担心在不同的模块中具有相同的类名可能会造成的问题。

注意:使用css-loader?modules 与css-loader的区别。

1. 使用css-loader

require('./Sub.less');

className = "div1"

1. 若使用css-loader?modules

import styles from './Greeter.css';

className={styles.root}

1. CSS预处理器

安装postcss-loader 和 autoprefixer（自动添加前缀的插件）

<https://segmentfault.com/q/1010000006987956>

错误修改: npm install webpack@1.12.9 --save-dev

1. 插件（Plugins）

插件（Plugins）是用来拓展Webpack功能的，它们会在整个构建过程中生效，执行相关的任务。Webpack有很多内置插件，同时也有很多第三方插件。

1. Loaders与Plugins的对比

loaders是在打包构建过程中用来处理源文件的（JSX，Scss，Less..），一次处理一个，插件并不直接操作单个文件，它直接对整个构建过程其作用。

1. 如何使用插件

要使用某个插件，我们需要通过npm安装它，然后要做的就是在webpack配置中的plugins关键字部分添加该插件的一个实例（plugins是一个数组）。

1. HtmlWebpackPlugin

这个插件的作用是依据一个简单的模板，帮你生成最终的Html5文件，这个文件中自动引用了你打包后的JS文件。每次编译都在文件名中插入一个不同的哈希值。

plugins: [new HtmlWebpackPlugin()]

这是最简单的写法，含义为在output输出路径下创建index.html文件，并引入打包后的js文件。

filename: 输出路径及输出文件名，默认是打包路径的index.html。

template: 模板文件路径。

title:标题名称(如果设置了template的title标题，则此设置不生效)。

参考: <http://www.cnblogs.com/haogj/p/5160821.html>

1. Hot Module Replacement

Hot Module Replacement（HMR）也是webpack里很有用的一个插件，它允许你在修改组件代码后，自动刷新实时预览修改后的效果。

注释:网页不刷新，组件重新渲染。

注意:①如果利用webpack-dev-server cli 并且加了--hot 这个选项，就不要在这里加入new webpack.HotModuleReplacementPlugin()这个插件了，否则会报Maximum call stack size exceeded错误。

<http://www.jianshu.com/p/976ca21c9245>

②.babelrc文件中的"env":{}键值对只在执行npm start进行本地服务器热加载时才添加，执行webpack或npm run build打包时不添加，否则报错。

1. 产品阶段的构建

在产品阶段，可能还需要对打包的文件进行额外的处理，比如说优化，压缩，缓存以及分离CSS和JS。这时候分解配置文件为多个小的文件可以使得事情井井有条。我们创建一个“webpack.production.config.js”的文件。去掉以下三部分内容:

devtool: 'eval-source-map'

new webpack.HotModuleReplacementPlugin()

devServer: {}

1. 优化插件

(1)OccurenceOrderPlugin :为组件分配ID，通过这个插件webpack可以分析和优先考虑使用最多的模块，并为它们分配最小的ID

(2)UglifyJsPlugin：压缩JS代码；

通过添加 compress: {warnings: false}键值对取消压缩时的警告。

(3)ExtractTextPlugin：分离CSS和JS文件

OccurenceOrder 和 UglifyJS plugins 都是内置插件，你需要做的只是安装它们。

报错解决: <http://www.tuicool.com/articles/bqequui>

<https://segmentfault.com/q/1010000006121117>

注释:①set NODE\_ENV=production的作用是使.babelrc中的"env": {}在production环境下不生效。

②new webpack.DefinePlugin({'process.env':{

'NODE\_ENV': JSON.stringify('production')

}

})的作用是处理编译后的错误信息:

React using a minified copy of the development build of React.

注意:这几个插件的使用可以大大压缩文件大小。

css-loader 和 style-loader的作用是通过执行打包后的js文件生成一个个小的style标签嵌入到head中，ExtractTextPlugin可以将打包之后的js文件有关样式的部分分离，生成一个link标签。

1. 缓存

缓存的最好方法是保证你的文件名和文件内容是匹配的（内容改变，名称相应改变），webpack可以把一个哈希值添加到打包的文件名中。用户会有合理的缓存了。

# 妙味课堂

1. 安装Node.js

<https://nodejs.org/en/>

命令行输入node -v查看node版本;

1. npm

NodeJS包管理和分发工具

http://npmjs.org/

命令行输入npm -v查看npm版本;

1. npm常用命令

npm init 创建package.json文件

npm install <module-name> -g/--save-dev/--save 安装模块

npm update <module-name > 更新模块

npm uninstall <module-name > 卸载模块

1. 操作步骤

新建文件夹并进入F:\mygit\webpack-react→shift+鼠标右键→点击打开命名窗口→npm init→重复回车(创建package.json文件)→npm install react --save-dev(将react模块作为依赖写入package.json)

注意: package.json文件不支持注释，否则打包失败。

1. webpack简介

一款模块加载器兼打包工具

支持AMD/CMD写法

处理依赖关系，然后解析出模块之间的依赖，将代码打包

把各种资源，都作为模块来使用和处理。

比如 js css Less Sass等。

<http://webpack.github.io/>

问题:如何安装webpack？

1. **安装**webpack

npm install webpack -g

安装后就在命令行中使用 webpack命令

把依赖写入 package.json

npm install webpack --save-dev

问题:如何使用webpack打包文件？

注释:npm install webpack --save-dev 不能够省略。

1. webpack打包命令

打包命令 webpack app.js output.js

app.js 打包的入口文件

output.js 打包后的文件

语法:webpack 需要打包的入口文件 打包生成文件

注释:打包入口文件及所在的文件夹是我们自己创建的，打包生成文件及所在的文件夹是自动生成的。

示例1:单个js文件的打包

步骤 F:\mygit\webpack-react\app中创建index.js→在此文件中书写js内容→webpack **app/index.js** **build/build.js**

问题：如何打包多个Js文件？

示例2:多个js文件的打包(入口文件只有一个)

步骤 F:\mygit\webpack-react\app中**创建**index.js和index1.js

→index1.js中书写以下内容:

module.exports='Hello';

→在index.js中书写以下内容(CMD写法):

var str=require('./index1.js');

document.body.innerHTML='<div>'+str+'</div>';

或者书写以下内容(AMD写法):

define(['./index1.js'],function(str){

document.body.innerHTML='<div>'+str+'</div>';

})

→webpack app/index.js build/build.js

注意:①改变文件内容之后需要重新打包。②CMD的写法更常用。③入口文件只有一个。

问题：如何打包CSS文件？

1. Webpack模块加载器(loader)

各种不同文件类型的资源, Webpack 有对应的模块 loader安装加载器。

npm install xxx-loader --save-dev

例如：css-loader style-loader 处理css文件和样式

更多参考：

<http://webpack.github.io/docs/using-loaders.html>

注意:①Webpack默认只能够处理js文件，如果需要处理css文件，需要使用模块加载器。

示例3:多个js文件和一个css文件的打包(入口文件只有一个)

步骤:npm install css-loader style-loader --sava-dev

→F:\mygit\webpack-react\app中创建index.js和index1.js、 style.css

→index1.js中书写以下内容:

module.exports='Hello';

→style.css中书写以下内容:

div{

color:red;

font-size: 60px;

}

→在index.js中书写以下内容(CMD写法):

var str=require('./index1.js');

require('style!css!./style.css');

document.body.innerHTML='<div>'+str+'</div>';

→webpack app/index.js build/build.js

问题：如何打包多个CSS文件？

示例4:多个js文件和多个css文件的打包(入口文件只有一个)

步骤: F:\mygit\webpack-react\app中创建index.js和index1.js、 style.css、reset.css

→index1.js中书写以下内容:

module.exports='Hello';

→reset.css中书写以下内容:

body{

background: gray;

}

→style.css中书写以下内容:

@import url(./reset.css);

div{

color:red;

font-size: 60px;

}

→在index.js中书写以下内容(CMD写法):

var str=require('./index1.js');

require('style!css!./style.css');

document.body.innerHTML='<div>'+str+'</div>';

→webpack app/index.js build/build.js

注释:①也可以在index.js中引用reset.css文件:

require('style!css!./reset.css');

注意:require('style!css!./reset.css');含义为使用style-loader和css-loader处理reset.css文件

1. Webpack配置项说明

entry: 打包的入口文件 String|Object

output: 配置打包结果 Object

path: 定义输出文件路径

Filename: 指定打包文件名称

Module: 定义了对模块的处理逻辑 Object

Loaders: 定义了一系列的加载器 Array

[{

test: 正则，匹配到的文件后缀名

loader/loaders:string|array，处理匹配到的文件

Include:String|Array 包含的文件夹

Exclude:String|Array 排除的文件夹(正则)

}]

resolve:{

extensions:['','.js',".css","jsx"]

//自动补全识别后缀优先级

}

1. Webpack配置步骤

打开F:\mygit\webpack-react文件夹→创建webpack.config.js文件(webpack配置文件)

//注释:由之前操作可知，没有webpack.config.js也可以打包

→在webpack.config.js文件中书写以下内容:

module.exports={

entry:'./app/index.js',

output:{

path:'./build/',

filename:'build.js',

},

module:{

loaders:[

{

test:/.css$/,

loaders:['style','css'],

exclude:'/node\_modules/'

}

]

},

resolve:{

extensions:['','.js',".css","jsx"]

}

}

→删除require('style!css!./style.css');中的style!css!

→在命令行(F:\mygit\webpack-react>)中直接输入webpack

注释:①只要文件改变就要重新打包。

②简化了打包的步骤:webpack app/index.js build/build.js命令简化为webpack命令。

③简化了css文件引入步骤:require('style!css!./style.css');简化为require('./style.css');

④文件引入时，可以省略文件后缀名。

问题:如何在服务器环境打开文件？

1. Webpack自动刷新功能

webpack-dev-server

轻量级的服务器

修改文件源码后，自动刷新页面就能把修改同步到页面上

安装webpack-dev-server

npm install webpack-dev-server -g

安装后在命令行中使用 webpack-dev-server命令

把依赖写入 package.json

npm install webpack-dev-server --save-dev

使用命令 webpack-dev-server --hot --inline 做到自动刷新

在webpack.config.js中配置服务

devServer: { }

参考:

<http://webpack.github.io/docs/webpack-dev-server.html#the-historyapifallback-option>

步骤:在命令行(F:\mygit\webpack-react>)中

→执行npm install webpack-dev-server -g

→执行npm install webpack-dev-server --save-dev

→index.html文件中<script src="./build/build.js"></script>

更改为:

<script src="http://localhost:3000/build.js"></script>

→执行webpack-dev-server --port 3000 --hot --inline

注释:①默认端口号为8080，如果执行webpack -dev-server发生错误时，可能是当前端口号被占用。可以通过webpack-dev-server --port 3000(新的端口号)方式解决。

问题:在服务器环境中，根目录中并没有build.js文件，src为什么修改为http://localhost:3000/build.js？

**答案：当使用webpack-dev-server服务进行自动刷新时，为了节约内存消耗，并不是每一次文件修改都重新进行打包，而只是将修改之后的文件保存在内存中，默认路径为根目录。因此，文件修改完成之后我们需要重新执行webpack命令进行打包，否则文件修改内容并不能够反映到本地build.js文件中。**

总结:http://localhost:3000/build.js为隐藏文件，./build/build.js才是真实文件，自动刷新功能利用了隐藏文件，修改完成之后需要执行**webpack**

命令将隐藏文件修改内容反映到真实文件上。

问题:如何简化webpack-dev-server --port 3000 --hot --inline命令。

1. 配置快捷键

步骤:打开package.json文件→在"scripts"属性值对象中添加→"build":"webpack-dev-server --port 3000 --hot --inline"键值对→npm run build

注释:此时npm run build命令等价于webpack-dev-server --port 3000 --hot --inline命令。

注意:除了webpack app/index.js build/build.js命令简化为webpack命令是在webpack.config.js文件中配置的之外，其他操作命令简化都是通过修改package.json中"scripts"进行配置的。即：webpack.config.js文件中设置的是webpack打包相关的配置，package.json文件中该设置的是**npm**命令相关的配置。

问题:如何在webpack中配置服务？

1. Webpack配置服务

步骤:打开webpack.config.js在module.exports={}对象中

→增加devServer: {historyApiFallback: true,hot: true,inline: true,progress: true}键值对。

问题:如何修改服务器默认访问路径并简化操作命令？

1. Webpack修改服务器访问路径

通过webpack-dev-server --content-base 相对路径，可以设置服务器访问路径。

步骤:打开package.json文件→在"scripts"属性值对象中添加→"build":"webpack-dev-server --port 3000 --hot --inline --content-base ./build/"键值对→npm run build

问题：webpack修改服务器访问路径后，如何在路径中自动生成html文件？

1. Webpack插件(html-webpack-plugin)

自动生成html文件并引入指定的js文件

安装：npm install html-webpack-plugin --save-dev

使用

在webpack.config.js中引入

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

在plugins中配置

plugins:[

new htmlWebpackPlugin({

title:"My first react app",

chunks:["index"]

})

]

步骤:执行npm install html-webpack-plugin --save-dev

→在webpack.config.js文件的首行(module.exports={}之前而不是之中)输入var htmlWebpackPlugin = require('html-webpack-plugin');

→在webpack.config.js文件中的module.exports={}对象中添加 plugins:[

new htmlWebpackPlugin({

title:"Welcome"

})

]

→webpack(此时在./build/目录中生成index.html文件)

→npm run build(此时在服务器路径中即可访问当前生成的html文件)

注释:①new htmlWebpackPlugin({})中的filename若未设置，则默认为index.html，chunks若未设置，则默认为引入全部output文件。②此时在./build/目录中生成index.html(默认文件名)文件，该文件中引入了build.js(output)文件(默认引入全部output文件)。

注意:该插件的作用是打包时，在output路径中创建html文件，并在该html文件中引入指定或默认output文件。

问题:webpack出入口(entry/output)文件如何设置多个？

1. Webpack设置多个入口文件

步骤:在app文件夹中创建abc.js→该文件内容为console.log(111);

→打开webpack.config.js在module.exports={}对象中出入口设置为

entry:{

'build':'./app/index.js',

'abc':'./app/abc.js'

},

output:{

path:'./build/',

filename:'[name].js',

}

→webpack(此时在build文件夹中生成两个文件:build.js与abc.js)

注释:①entry对象中的键为打包将生成文件的文件名。②此时在./build/目录(**output中的path路径**)中生成的index.html(默认文件名)文件，并引入了build.js和abc.js两个(output)文件(默认引入全部output文件)。

问题:如何自动生成多个html文件并且指定output文件

1. Webpack为自动生成的html文件指定output文件

→打开webpack.config.js在module.exports={}对象中修改

→plugins:[

new htmlWebpackPlugin({

title:"welcome",

chunks:['build']

}),

new htmlWebpackPlugin({

title:"欢迎",

filename:'class.html',

chunks:['abc']

})

]

→webpack

注释:①new htmlWebpackPlugin({})中的filename若未设置，则默认为index.html，chunks若未设置，则默认为引入全部output文件。②此时在./build/目录中生成的index.html文件和class.html文件，其中index.html文件中引入了build.js(output)文件，class.html文件中引入了abc.js(output)文件。

问题:如何指定webpack配置文件并简化操作命令？

1. Webpack配置文件名修改

webpack.config.js文件复制一份存放在相同的根路径下

→文件名修改为webpack.html.config.js

→打开package.js文件，在script对象中添加以下键值对

→"start\_html":"webpack --config webpack.html.config.js"

→npm run start\_html(使用webpack.html.config.js配置文件打包)

注释:此时执行webpck命令，则使用webpack.config.js默认配置文件打包；执行npm run start\_html命令，则使用webpack.html.config.js配置文件进行打包。

问题:如何将ES6语法代码转化为ES5语法代码？

1. babel---转码器

babel作用

将ES6代码转为ES5代码

官网：http://babeljs.io/

安装babel-cli

npm install babel-cli -g

安装后就在命令行中使用 babel 命令

把依赖写入 package.json

npm install babel-cli --save-dev

转换命令：babel app.js --out-file build.js

使用es2015

npm install --save-dev babel-preset-es2015

在目录下创建.babelrc文件，设置为

{"presets": ["es2015"]}

步骤:根目录下创建文件夹es6→该文件夹中创建index.js

→webpack.html.config.js复制一份放在相同跟目录下，重命名为webpack.es6.config.js

→该配置文件中修改

entry:{

'build':'./es6/index.js',

},

output:{

path:'./es6\_build/',

filename:'[name].js',

},

plugins:[

new htmlWebpackPlugin({

title:"welcome",

chunks:['build']

})

]

→打来package.json文件，在script对象中添加以下键值对

→"start\_es6":"webpack --config webpack.es6.config.js"

→执行npm run start\_es6命令

//注释:以上所有操作(设置entry入口文件，设置output出口位置，自动生成html文件，指定webpack打包配置文件，简化打包操作指令等)都是webpack+babel操作的准备工作。

→npm install babel-cli -g

→npm install babel-cli --save-dev

→npm install --save-dev babel-preset-es2015

→在目录下创建.babelrc文件，设置为

{"presets": ["es2015"]}

→在es6/index.js文件中书写以下代码

var fn = ()=>{

consolg.log(123);

}

fn();

→babel es6/index.js --out-file a.js

注释:此时在根目录下自动生成a.js文件，该文件内容为:

"use strict";

var fn = function fn() {

consolg.log(123);

};

fn();

1. webpack+babel

安装babel-loader加载器

使用webpack处理文件中ec6语法

在webpack.config.js中加入对应的处理

{

test: /\.js$/

loader：'babel',

query:['es2015']

}

步骤:→npm install babel-loader -g

→npm install babel-loader --save-dev

→打开webpack.es6.config.js文件

→在第二行(module.exports之前)添加path模块

var path=require('path');

//注释:path模块为内置模块，不需要手动安装

→在module:{loaders:[]}数组中添加以下加载器

{

test: /.js$/,

loaders:['babel-loader'],

exclude:'/node\_modules/',

include:path.resolve(\_\_dirname,'/es6/')

}

//注释:include:path.resolve(\_\_dirname,'/es6/')的含义为**只在当 前路径下的es6文件夹下寻找.js为后缀的匹配文件**

→npm run start\_es6

→打开package.json文件，在"scripts"对象中增加以下键值对

"build\_es6": "webpack-dev-server --port 3100 --hot --inline --config webpack.es6.config.js"

→npm run build\_es6

注释:由于4000端口号已被占用，当前使用3100端口号。

1. 待学习...

# Webpack傻瓜式指南

<https://zhuanlan.zhihu.com/p/20367175>

<https://zhuanlan.zhihu.com/p/20397902>

<https://zhuanlan.zhihu.com/p/20522487>

1. webpack的定义

MODULE BUNDLER(模块打包工具)

把有依赖关系的各种文件打包成一系列的静态资源。

1. Path

var path = require('path');

var ROOT\_PATH = path.resolve(\_\_dirname);

var APP\_PATH = path.resolve(ROOT\_PATH, 'app');

var BUILD\_PATH = path.resolve(ROOT\_PATH, 'build');

1. Entry入口的另一种写法

'ab': [APP\_PATH+'/a.js', APP\_PATH + '/b.js']

含义为将a.js与b.js打包成ab.js

1. 添加CSS样式

css-loader会遍历css文件，找到所有的url(...)并且处理。style-loader会把所有的样式插入到你**页面的一个style tag中**。

loaders: ['style', 'css']

注意loaders的处理顺序是从右到左的，这里就是先运行css-loader然后是style-loader。

1. Less

npm install less less-loader --save-dev

1. 处理图片和其他静态文件

根据需求将一些图片自动转成base64编码，减轻网络请求。

npm install url-loader --save-dev

npm install file-loader --save-dev

loader:'url?limit=40000&name=images/[hash:8].[name].[ext]'含义为图片小于40000Byte时， 以base64编码的方式引用，图片大于40000Byte时，正常打包。 name 字段指定了在打包根目录（output.path）下生成名为 images 的文件夹，并在原图片名前加上8位 hash 值。

注意:图片大小大于40000Byte，路径错误，如何解决？

答案:完善output.publicPath

output: {publicPath: '../../dist/'}

publicPath的值为当前环境(本地/服务器)中输出的html页面地址到打包地址dist的相对路径。

1. 添加第三方库

npm install jquery --save-dev

var $ = require('jquery'); //不可以省略

注意:npm install jquery --save-dev之后，jquery是全局的，但$并未指向jquery。

1. 配置webpack-dev-server代理
2. 使用preLoaders和postLoaders
3. 加载jQuery plugin或其他第三方库

方法一:使用webpack.ProvidePlugin,把全局变量插入到所有代码中。

方法二:使用imports-loader。

1. 分离项目代码和第三方库

(1)修改entry入口文件

entry: {

app: path.resolve(APP\_PATH, 'index.js'),

//添加要打包在vendors里面的库

vendors: ['jquery', 'moment']

},

1. 添加CommonsChunkPlugin

plugins: [

//把入口文件里面的数组打包成verdors.js

new webpack.optimize.CommonsChunkPlugin('vendors', 'vendors.js'),

]

1. 配置Babel让React支持ES6

npm install babel-loader --save-dev

npm install babel-preset-es2015 babel-preset-react --save-dev

这里安装了babel的主体和两个babel的preset, 你可以把preset看成一个包，里面有各种各样一系列的插件。babel-preset-es2015为babel的es6语法包，babel-preset-react为babel的react语法包

1. 添加React Transform支持

React Hot Loading已经过时了，开发者也宣布已经停止维护，现在有一个更强大的babel plugin： React Transform来代替他。

(与《[入门Webpack，看这篇就够了](http://www.jianshu.com/p/42e11515c10f)》配置方式不同)

1. npm install --save-dev babel-plugin-react-transform
2. npm install --save-dev react-transform-hmr
3. npm install --save-dev react-transform-catch-errors redbox-react
4. npm install babel-preset-react-hmre --save-dev
5. {

"presets": ["react", "es2015"],

//在开发的时候才启用HMR和Catch Error

"env": {

"development": {

"presets": ["react-hmre"]

}

}

}

# 一小时包教会 - webpack 入门指南

<http://www.w2bc.com/Article/50764>

1. 什么是 webpack

webpack是一款模块加载器兼打包工具，它能把各种资源，例如JS(含JSX)、coffee、样式(含less/sass)、图片等都作为模块来使用和处理。

1. 我们可以直接使用 require() 的形式来引入各模块，即使它们可能需要经过编译(比如JSX和sass)，webpack 有着各种健全的加载器(loader)来处理这些模块。
2. webpack官网

<http://webpack.github.io/>

文档地址

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

1. webpack 的优势
2. 对AMD/CMD 的支持全面，方便旧项目进行代码迁移;
3. 各种资源都可以被模块化;
4. 被模块化;
5. 扩展性强，插件机制完善( react-hot-loader);
6. 使用webpack的CommonsChunkPlugin提取公共代码的3种方式

<http://blog.csdn.net/github_26672553/article/details/52280655>

1. plugins 是插件项，这里我们使用了一个 CommonsChunkPlugin 的插件，它用于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。
2. 在entry中，同一个js文件不能够作为两个入口。

<https://github.com/webpack/webpack/issues/300>

1. CommonsChunkPlugin与 HtmlWebpackPlugin两个插件共同使用时的报错处理

<http://www.tuicool.com/articles/7rYFvqJ>

1. resolve.root

<http://blog.csdn.net/keliyxyz/article/details/51527476>

包含你的模块的目录（绝对路径）。也可能是目录的数组。需要将单个目录添加到搜索路径的情况下，才用这个设置，查找module的话从这里开始查找。

一个目录数组。这个目录将解析给当前目录以及它的祖先目录，在这里查找模块。它的功能类似于 node 的 node\_modules目录。默认值为：["web\_modules", "node\_modules"]

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

The directory (absolute path) that contains your modules. May also be an array of directories. This setting should be used to add individual directories to the search path.

It must be an absolute path! Don’t pass something like ./app/modules.

1. 运行 webpack

$ 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

其中的 -p 是很重要的参数，曾经一个未压缩的 700kb 的文件，压缩后直接降到 180kb（主要是样式这块一句就独占一行脚本，导致未压缩脚本变得很大）。

1. 脚本引入

直接在页面引入 webpack 最终生成的页面脚本即可:

<script src="dist/js/page/common.js"></script>

<script src="dist/js/page/index.js"></script>

连样式都不用引入，脚本执行时会动态生成<style>并标签打到head里。

1. 独立打包样式文件

时候可能希望项目的样式能不要被打包到脚本中，而是独立出来作为.css，然后在页面中以<link>标签引入。这时候我们需要 extract-text-webpack-plugin。

注释: 只使用style-loader和css-loader的时候css样式设置被打包进js文件中，并且通过style标签插入到head头部;

使用extract-text-webpack-plugin的时候，css样式设置打包时从js文件中分离，并且通过link标签链入到head标签中，且一个出口文件生成一个link标签。

# Webpack中文指南

http://webpackdoc.com/index.html

1. webpack的官网: http://webpack.github.io/

文档地址: http://webpack.github.io/docs/

1. Webpack 是当下最热门的前端资源模块化管理和打包工具。它可以将许多松散的模块按照依赖和规则打包成符合生产环境部署的前端资源。还可以将按需加载的模块进行代码分隔，等到实际需要的时候再异步加载。通过 loader 的转换，任何形式的资源都可以视作模块，比如 CommonJs 模块、 AMD 模块、 ES6 模块、CSS、图片、 JSON、Coffeescript、 LESS 等。
2. 前端开发和其他开发
3. 首先是前端是基于多语言、多层次的编码和组织工作。
4. 其次前端产品的交付是基于浏览器，这些资源是通过增量加载的方式运行到浏览器端。

为了在开发环境组织好这些碎片化的代码和资源，并且保证他们在浏览器端快速、优雅的加载和更新，就需要一个模块化系统。模块系统主要解决模块的定义、依赖和导出。

1. 若干个<script>标签的模块系统的缺点
2. 全局作用域下容易造成变量冲突。
3. 只能按照 <script> 的书写顺序进行加载。
4. 开发人员必须主观解决模块和代码库的依赖关系。
5. 资源难以管理，导致代码库混乱。
6. CommonJS规范

服务器端的 Node.js 遵循 CommonJS规范，该规范的核心思想是允许模块通过 require 方法来同步加载所要依赖的其他模块，然后通过 exports 或 module.exports 来导出需要暴露的接口。

优点: (1)服务器端模块便于重用

(2)NPM 中已经有将近20万个可以使用模块包

1. 简单并容易使用

缺点: (1)同步的模块加载方式不适合在浏览器环境中，同步意味着阻塞加载，浏览器资源是异步加载的。

1. 不能非阻塞的并行加载多个模块

实现: Node.js、Browserify、modules-webmake

1. AMD

优点: (1)适合在浏览器环境中异步加载模块

(2)可以并行加载多个模块

缺点: (1)代码的阅读和书写比较困难，模块定义方式的语义不顺畅

(2)不符合通用的模块化思维方式，是一种妥协的实现

实现: RequireJS、curl

1. CMD

优点: (1)依赖就近，延迟执行

(2)可以很容易在 Node.js 中运行

缺点: (1)依赖 SPM 打包，模块的加载逻辑偏重

实现: Sea.js、coolie

1. ES6 模块

EcmaScript6 标准增加了 JavaScript 语言层面的模块体系定义。ES6 模块的设计思想，是尽量的静态化，使得编译时就能确定模块的依赖关系，以及输入和输出的变量。CommonJS 和 AMD 模块，都只能在运行时确定这些东西。

优点: (1)容易进行静态分析

(2)面向未来的 EcmaScript 标准

缺点: (1)原生浏览器端还没有实现该标准

(2)全新的命令字，新版的 Node.js才支持

实现: Babel

1. 期望的模块系统

可以兼容多种模块风格，尽量可以利用已有的代码，不仅仅只是 JavaScript 模块化，还有 CSS、图片、字体等资源也需要模块化。

1. 前端模块加载
2. 每个模块文件都单独请求

请求次数过多，导致应用启动速度慢。

1. 模块打包成一个文件然后只请求一次

流量浪费、初始化过程慢。

1. 分块传输，按需进行懒加载

在实际用到某些模块的时候再增量更新。

要实现模块的按需加载，就需要一个对整个代码库中的模块进行静态分析、编译打包的过程。

1. 所有资源都是模块

前端开发过程中还涉及到样式、图片、字体、HTML 模板等等众多的资源。这些资源还会以各种方言的形式存在，比如 coffeescript、 less、 sass、众多的模板库、多语言系统（i18n）等等。

如果他们都可以视作模块，并且都可以通过require的方式来加载，将带来优雅的开发体验，比如：

require("./style.css");

require("./style.less");

require("./template.jade");

require("./image.png");

那么如何做到让 require 能加载各种资源呢？

1. 静态分析

在编译的时候，要对整个代码进行静态分析，分析出各个模块的类型和它们依赖关系，然后将不同类型的模块提交给适配的加载器来处理。比如一个用 LESS 写的样式模块，可以先用 LESS 加载器将它转成一个CSS 模块，在通过 CSS 模块把他插入到页面的 <style> 标签中执行。

同时，为了能利用已经存在的各种框架、库和已经写好的文件，我们还需要一个模块加载的兼容策略，来避免重写所有的模块。

1. Webpack 是一个模块打包器。它将根据模块的依赖关系进行静态分析，然后将这些模块按照指定的规则生成对应的静态资源。
2. Loader

Webpack 本身只能处理原生的 JavaScript 模块，但是 **loader 转换器可以将各种类型的资源转换成 JavaScript 模块。**这样，任何资源都可以成为 Webpack 可以处理的模块。

Loader 可以理解为是模块和资源的转换器，它本身是一个函数，接受源文件作为参数，返回转换的结果。这样，我们就可以通过 require 来加载任何类型的模块或文件。

1. Node.js 自带了软件包管理器 npm，Webpack 需要 Node.js v0.6 以上支持。
2. loader的部分特点

Loader 可以通过管道方式链式调用，每个 loader 可以把资源转换成任意格式并传递给下一个 loader ，但是最后一个 loader 必须返回 JavaScript。

Loader 可以同步或异步执行。

Loader 可以通过 npm 发布和安装。

除了通过 package.json 的 main 指定，通常的模块也可以导出一个 loader 来使用。

插件可以让 loader 拥有更多特性。

1. 开发环境

当项目逐渐变大，webpack 的编译时间会变长，可以通过参数让编译的输出内容带有进度和颜色。

webpack --progress --colors

如果不想每次修改模块后都重新编译，那么可以启动监听模式。开启监听模式后，没有变化的模块会在编译后缓存到内存中，而不会每次都被重新编译，所以监听模式的整体速度是很快的。

webpack --progress --colors --watch

当然，使用 webpack-dev-server 开发服务是一个更好的选择。它将在 localhost:8080 启动一个 express 静态资源 web 服务器，并且会以监听模式自动运行 webpack。

1. 故障处理

Webpack 的配置比较复杂，很容出现错误，下面是一些通常的故障处理手段。一般情况下，webpack 如果出问题，会打印一些简单的错误信息，比如模块没有找到。我们还可以通过参数 --display-error-details 来打印错误详情。

webpack --display-error-details