# 命令行指令作用： npm run dev 原因：

npm run XXX是执行配置在package.json中的脚本，比如：

"scripts": {

"dev": "node build/dev-server.js",

"build": "node build/build.js",

"unit": "karma start test/unit/karma.conf.js --single-run",

"e2e": "node test/e2e/runner.js",

"test": "npm run unit && npm run e2e",

"lint": "eslint --ext .js,.vue src test/unit/specs test/e2e/specs"

},

只有这里配置了，你才能run，所以不是所有的项目都能npm run dev/build。要了解这些命令做了什么，就要去scripts中看具体执行的是什么代码。这里就像是一些命令的快捷方式，免去每次都要输入很长的的命令（比如unit那行）。

为什么会出现ERROR，就是因为在跑这些对应的脚本文件的时候，可能是某些依赖没有被加载等的。

一般项目都会有build, dev, unit等，从名字上基本能看出来是干什么的。比如上面配置的unit，就是开启karma去跑单元测试，具体测试内容，要去看karma.conf.js；e2e就是end to end的端到端测试；而test则会将单元测试和端到端测试都执行。

# 1.webpack 起步

全局安装webpack

npm install webpack -g

**1.cats.js**

var cats = ['dave', 'henry', 'martha'];

module.exports = cats;

**2.app.js (Entry Point)**

cats = require('./cats.js');

console.log(cats);

The “entry point” is where your application will start, and where webpack will start tracking dependencies between modules.

Give webpack the entry point (app.js) and specify an output file (app.bundle.js):

3.运行webpack 指定输入输出文件名

webpack ./app.js app.bundle.js

4.用node 运行bundle文件

node app.bundle.js

["dave", "henry", "martha"]

1.开始构建Npm项目

npm init *# (answer the questions)*

2.构建webpack 依赖

npm install --save-dev webpack

3.创建配置文件webpack.config.js:

module.exports = {

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

output: {

path: './bin',//[output.path](http://www.css88.com/doc/webpack2/concepts/output/" \l "output-path) 对应一个**绝对路径**，此路径是你希望一次性打包的目录。

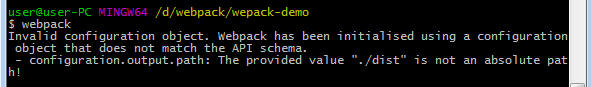
filename: 'app.bundle.js'

}

};

4.直接在命令行运行webpack

Webpack

实际操作报错：

‘./dist’不是绝对路径；

解决方法：output.path更改为/webpack/wepack-demo/dist >>文件目录为D:\webpack\wepack-demo

5. 运行bin/app.bundle.js

node bin/app.bundle.js

["dave", "henry", "martha"]

# webpack装载机应用 loader

webpack only supports JavaScript modules natively, but most people will be using a transpiler for ES2015, CoffeeScript, TypeScript, etc. They can be used in webpack by using [loaders](https://webpack.github.io/docs/using-loaders.html" \o "Using Loaders).

Loaders are special modules webpack uses to ‘load’ other modules (written in another language) into JavaScript (that webpack understands). For example, [babel-loader](https://github.com/babel/babel-loader) uses Babel to load ES2015 files.

1. Install Babel and the presets:

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

1. Install babel-loader:

npm install --save-dev babel-loader

1. Configure Babel to use these presets by adding .babelrc

{ "presets": [ "es2015" ] }

1. Modify webpack.config.js to process all .js files using babel-loader.

module.exports = {

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

output: {

path: './bin',

filename: 'app.bundle.js',

},

module: {

loaders: [{

test: /\.js$/,

exclude: /node\_modules/,

loader: 'babel-loader'

}]

}

}

*We are excluding node\_modules here because otherwise all external libraries will also go through Babel, slowing down compilation.*

1. Install the libraries you want to use (in this example, jQuery):

npm install --save jquery babel-polyfill

*We are using --save instead of --save-dev this time, as these libraries will be used in runtime. We also use babel-polyfill so that ES2015 APIs are available in older browsers.*

1. Edit src/app.js:

import 'babel-polyfill';

import cats from './cats';

import $ from 'jquery';

$('<h1>Cats</h1>').appendTo('body');

const ul = $('<ul></ul>').appendTo('body');

for (const cat of cats) {

$('<li></li>').text(cat).appendTo(ul);

}

1. Bundle the modules using webpack:

webpack

1. Add index.html so this app can be run in browser:

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

</head>

<body>

<script src="bin/app.bundle.js" charset="utf-8"></script>

</body>

</html>

# 3.webpack 插件

Usually you’ll want to do some additional processing of the bundle in your workflow. An example would be minifying your file so that clients can load it faster. This can be done with [plugins](https://webpack.github.io/docs/using-plugins.html" \o "Using Plugins). We’ll add the uglify plugin to our configuration:

const webpack = require('webpack');

module.exports = {

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

output: {

path: './bin',

filename: 'app.bundle.js',

},

module: {

loaders: [{

test: /\.js?$/,

exclude: /node\_modules/,

loader: 'babel-loader',

}]

},

plugins: [

**new** webpack.optimize.UglifyJsPlugin({

compress: {

warnings: false,

},

output: {

comments: false,

},

}),

]

}

The Uglify plugin is included with webpack so you don’t need to add additional modules, but this may not always be the case. You can write your own [custom plugins](https://webpack.github.io/docs/how-to-write-a-plugin.html" \o "How to write a Plugin). For this build, the uglify plugin cut the bundle size from 1618 bytes to 308 bytes.

# webpack watch文件变化。自动编译

If the project grows the compilation may take a bit longer. So we want to display some kind of progress bar. And we want colors…

1. We can achieve this with

webpack --progress --colors

1. WATCH MODE
2. We don’t want to manually recompile after every change…

webpack --progress --colors --watch

Webpack can cache unchanged modules and output files between compilations.

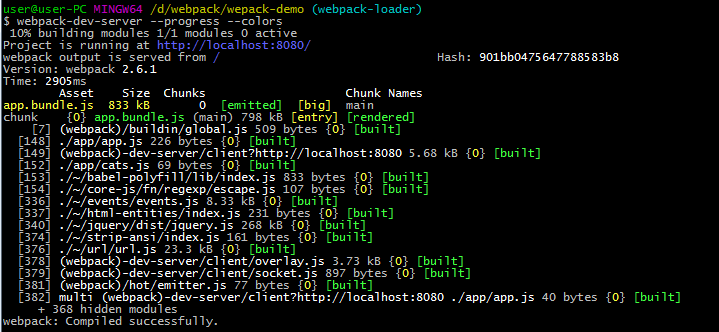
*When using watch mode, webpack installs file watchers to all files, which were used in the compilation process. If any change is detected, it’ll run the compilation again. When caching is enabled, webpack keeps each module in memory and will reuse it if it isn’t changed.*

1. DEVELOPMENT SERVER

The development server is even better.

npm install webpack-dev-server -g

webpack-dev-server --progress --colors



Webpack-serve 是自动监测变化并更新http://localhost:8080/

****NOTE:**** The webpack dev server is a separate NPM package. You can install it with: npm install webpack-dev-server.

Content Base

The webpack-dev-server will serve the files in the current directory, unless you configure a specific content base.

$ webpack-dev-server --content-base build/

Using this configuration, webpack-dev-server will serve the static files in your build folder. It’ll watch your source files, and recompile the bundle whenever they are changed.

This modified bundle is served from memory at the relative path specified in publicPath (see [API](http://webpack.github.io/docs/webpack-dev-server.html" \l "api)). It will not be written to your configured output directory. Where a bundle already exists at the same URL path, the bundle in memory takes precedence (by default).

Using the configuration above, the bundle is available at localhost:8080/assets/bundle.js.

To load your bundled files, you will need to create an index.html file in the build folder from which static files are served (--content-base option).

By default, go to localhost:8080/ to launch your app. In this example’s configuration (with publicPath), go to localhost:8080/assets/.

自动刷新Automatic Refresh

The webpack-dev-server supports multiple modes to automatically refresh the page:

* ****Iframe mode**** (page is embedded in an iframe and reloaded on change)
* ****Inline mode**** (a small webpack-dev-server client entry is added to the bundle which refreshes the page on change)

这里只看inline mode

### Inline mode

To use the inline mode, either

* specify --inline on the command line.
* specify devServer: { inline: true } in your webpack.config.js

*webpack-dev-middleware 是dev-serve的优化*

*Note: The webpack-dev-middleware is for advanced users. See [webpack-dev-server](http://webpack.github.io/docs/webpack-dev-server.html) for a ready-to-use solution.  
Note: For multiple webpack [configuration](http://webpack.github.io/docs/configuration.html) see [middware](https://github.com/hinell/webpack-middware) or [dev-middleware](https://github.com/kriasoft/webpack-middleware)   
If you are looking for this middleware for [koa2](https://github.com/koajs/koa/tree/v2.x) ecosystem loaded with HMR support see at [koa-webpack-middleware](https://github.com/leecade/koa-webpack-middleware).*

You can use it in two modes:

* watch mode (default): The compiler recompiles on file change.
* lazy mode: The compiler compiles on every request to the entry point.

# 5.代理：Proxy

The Webpack dev server makes use of [http-proxy-middleware](https://github.com/chimurai/http-proxy-middleware) to optionally proxy requests to a separate, possibly external, backend server. A sample configuration is below.

proxy: {

'/api': {

target: 'https://other-server.example.com',

secure: false

}

}

*// In webpack.config.js*

{

devServer: {

proxy: {

'/api': {

target: 'https://other-server.example.com',

secure: false

}

}

}

}

*// Multiple entry*

proxy: [

{

context: ['/api-v1/\*\*', '/api-v2/\*\*'],

target: 'https://other-server.example.com',

secure: false

}

]

See the [http-proxy-middleware Options documentation](https://github.com/chimurai/http-proxy-middleware" \l "options) for available configuration.

Proxying some URLs can be useful for a variety of configurations. One example is to serve JavaScript files and other static assets from the local development server but still send API requests to an external backend development server. Another example is splitting requests between two separate backend servers such as an authentication backend and a application backend.

### Bypass the Proxy

(Added in v1.13.0) The proxy can be optionally bypassed based on the return from a function. The function can inspect the HTTP request, response, and any given proxy options. It must return either false or a URL path that will be served instead of continuing to proxy the request.

For example, the configuration below will not proxy HTTP requests that originate from a browser. This is similar to the historyApiFallback option: browser requests will receive the HTML file as normal but API requests will be proxied to the backend server.

proxy: {

'/some/path': {

target: 'https://other-server.example.com',

secure: false,

bypass: **function**(req, res, proxyOptions) {

**if** (req.headers.accept.indexOf('html') !== -1) {

console.log('Skipping proxy for browser request.');

**return** '/index.html';

}

}

}

### Rewriting URLs of proxy request

(Added in v1.15.0) The request to the proxy can be optionally rewritten by providing a function. The function can inspect and change the HTTP request.

For example, the configuration below will rewrite the HTTP requests to remove the part /api at the beginning of the URL.

proxy: {

'/api': {

target: 'https://other-server.example.com',

pathRewrite: {'^/api' : ''}

}

}

Note that pathRewrite is a feature from http-proxy-middleware, so check out [their docs](https://github.com/chimurai/http-proxy-middleware" \l "http-proxy-middleware-options) for more configuration.

### Proxying local virtual hosts

It seems that http-proxy-middleware pre-resolves the local hostnames into localhost, you will need the following config to fix the proxy request:

**var** server = **new** webpackDevServer(compiler, {

quiet: false,

stats: { colors: true },

proxy: {

"/api": {

"target": {

"host": "action-js.dev",

"protocol": 'http:',

"port": 80

},

ignorePath: true,

changeOrigin: true,

secure: false

}

}

});

server.listen(8080);