**[BABEL-LOADER配置](https://www.cnblogs.com/MrZWJ/p/11697067.html)**

1、npm i @babel/core、@babel/preset-env、@babel/runtime、@babel/plugin-transform-runtime、@babel/plugin-proposal-class-properties、@babel/preset-react -D

2、然后在.babelrc文件中添加{"presets":["@babel/preset-env","@babel/preset-react"], "plugins":["@babel/transform-runtime","@babel/plugin-proposal-class-properties"]}

3、在webpackconfig.js文件中添加

module: {

        rules: [

            {

              test: /\.js$/,

              exclude: /(node\_modules|bower\_components)/,

              use: {

                loader: 'babel-loader',

                options: {

                  presets: ['@babel/preset-env']

                }

              }

            }

          ]

    }

4、npm run dev 大功告成啦

**阅读目录**

[一：理解 babel之配置文件.babelrc 基本配置项](https://www.cnblogs.com/tugenhua0707/p/9452471.html" \l "_labe1_0)

[二：在webpack中配置babel](https://www.cnblogs.com/tugenhua0707/p/9452471.html#_labe1_1)

[回到顶部](https://www.cnblogs.com/tugenhua0707/p/9452471.html#_labelTop)

**一：理解 babel之配置文件.babelrc 基本配置项**

**1. 什么是babel? 它是干什么用的？**

   ES6是2015年发布的下一代javascript语言标准，它引入了新的语法和API，使我们编写js代码更加得心应手，比如class，let,for...of promise等等这样的，但是可惜的是这些js新特性只被最新版本的浏览器支持，但是低版本浏览器并不支持，那么低版本浏览器下就需要一个转换工具，把es6代码转换成浏览器能识别的代码，babel就是这样的一个工具。可以理解为 babel是javascript语法的编译器。

**2. Babel编译器**  
在Babel执行编译的过程中，会从项目的根目录下的 .babelrc文件中读取配置。.babelrc是一个json格式的文件。  
在.babelrc配置文件中，主要是对预设(presets) 和 插件(plugins) 进行配置。.babelrc配置文件一般为如下：

[复制代码](javascript:void(0);)

{

"plugins": [

[

"transform-runtime",

{

"polyfill": false

}

]

],

"presets": [

[

"env",

{

"modules": false

}

],

"stage-2",

"react"

]

}

[复制代码](javascript:void(0);)

**2.1 plugins**  
该属性是告诉babel要使用那些插件，这些插件可以控制如何转换代码。

**1. 理解 babel-polyfill 和 babel-runtime 及 babel-plugin-transform-runtime**

   Babel默认只转换新的javascript语法，而不转换新的API，比如 Iterator, Generator, Set, Maps, Proxy, Reflect,Symbol,Promise 等全局对象。以及一些在全局对象上的方法(比如 Object.assign)都不会转码。  
比如说，ES6在Array对象上新增了Array.form方法，Babel就不会转码这个方法，如果想让这个方法运行，必须使用 babel-polyfill来转换等。

因此：babel-polyfill和babel-runtime就是为了解决新的API与这种全局对象或全局对象方法不足的问题，因此可以使用这两个插件可以转换的。

**那么他们两者的区别是什么？**  
**babel-polyfill** 的原理是当运行环境中并没有实现的一些方法，babel-polyfill会做兼容。  
**babel-runtime** 它是将es6编译成es5去执行。我们使用es6的语法来编写，最终会通过babel-runtime编译成es5.也就是说，不管浏览器是否支持ES6，只要是ES6的语法，它都会进行转码成ES5.所以就有很多冗余的代码。

**babel-polyfill** 它是通过向全局对象和内置对象的prototype上添加方法来实现的。比如运行环境中不支持Array.prototype.find 方法，引入polyfill, 我们就可以使用es6方法来编写了，但是缺点就是会造成全局空间污染。

**babel-runtime:** 它不会污染全局对象和内置对象的原型，比如说我们需要Promise，我们只需要import Promise from 'babel-runtime/core-js/promise'即可，这样不仅避免污染全局对象，而且可以减少不必要的代码。

虽然 babel-runtime 可以解决 babel-polyfill中的避免污染全局对象，但是它自己也有缺点的，比如上，如果我现在有100个文件甚至更多的话，难道我们需要一个个文件加import Promise from 'babel-runtime/core-js/promise' 吗？那这样肯定是不行的，因此这个时候出来一个 叫 babel-plugin-transform-runtime，  
它就可以帮助我们去避免手动引入 import的痛苦，并且它还做了公用方法的抽离。比如说我们有100个模块都使用promise，但是promise的polyfill仅仅存在1份。  
这就是 babel-plugin-transform-runtime 插件的作用。

**2. 理解 babel-plugin-transform-runtime 的配置一些选项**

因此通过上面的理解，我们可以对 transform-runtime 通过如下配置 plugins; 如下代码：

[复制代码](javascript:void(0);)

{

'plugins': [

[

'transform-runtime',

{

'helpers': false,

'polyfill': false,

'regenerator': true,

'moduleName': 'babel-runtime'

}

]

]

}

[复制代码](javascript:void(0);)

配置项可以看官网，[查看官网](http://babeljs.io/docs/en/babel-plugin-transform-runtime/" \l "helpers" \t "_blank)

**helpers: 默认值为true，**表示是否开启内联的babel helpers(即babel或者环境本来存在的某些对象方法函数)如：extends，etc这样的  
在调用模块名字时将被替换名字。

**polyfill：默认值为true**，表示是否把内置的东西(Promise, Set, Map)等转换成非全局污染的。

**regenerator：默认值为true，**是否开启generator函数转换成使用regenerator runtime来避免污染全局域。

**moduleName：默认值为 babel-runtime，**当调用辅助 设置模块（module）名字/路径.  
比如如下这样设置：

{

"moduleName": "flavortown/runtime"

}

import引入文件如下这个样子：

import extends from 'flavortown/runtime/helpers/extends';

**3 presets**  
presets属性告诉Babel要转换的源码使用了哪些新的语法特性，presets是一组Plugins的集合。

**3.1 理解 babel-preset-env**

比如：

**babel-preset-es2015: 可以将es6的代码编译成es5.**  
**babel-preset-es2016: 可以将es7的代码编译为es6.**  
**babel-preset-es2017: 可以将es8的代码编译为es7.**  
**babel-preset-latest: 支持现有所有ECMAScript版本的新特性。**

举个列子，比如我们需要转换es6语法，我们可以在 .babelrc的plugins中按需引入一下插件，比如：  
check-es2015-constants、es2015-arrow-functions、es2015-block-scoped-functions等等几十个不同作用的plugin：  
那么配置项可能是如下方式：

[复制代码](javascript:void(0);)

// .babelrc

{

"plugins": [

"check-es2015-constants",

"es2015-arrow-functions",

"es2015-block-scoped-functions",

// ...

]

}

[复制代码](javascript:void(0);)

但是Babel团队为了方便，将同属ES2015的几十个Transform Plugins集合到babel-preset-es2015一个Preset中，这样我们只需要在.babelrc的presets加入es2015一个配置就可以完成全部ES2015语法的支持了：  
如下配置：

[复制代码](javascript:void(0);)

// .babelrc

{

"presets": [

"es2015"

]

}

[复制代码](javascript:void(0);)

但是我们随着时间的推移，将来可能会有跟多的版本插件,比如 bebel-preset-es2018,.... 等等。  
因此 babel-preset-env 出现了，它的功能类似于 babel-preset-latest，它会根据目标环境选择不支持的新特性来转译。

首先需要在项目中安装，如下命令：

npm install babel-preset-env --save-dev

在.babelrc配置文件中 可以如下简单的配置：

{

"presets": ['env']

}

我们还可以仅仅配置项目所支持的浏览器的配置

1. 支持每个浏览器最后两个版本和safari大于等于7版本所需的polyfill代码转换，我们可以如下配置：

[复制代码](javascript:void(0);)

{

'presets': [

['env', {

'target': {

'browsers': ['last 2 versions', 'safari >= 7']

}

}]

]

}

[复制代码](javascript:void(0);)

2. 支持市场份额超过5%的浏览器，可以如下配置：

[复制代码](javascript:void(0);)

{

'presets': [

['env', {

'target': {

'browsers': '> 5%'

}

}]

]

}

[复制代码](javascript:void(0);)

3. 指定浏览器版本，可以如下配置：

[复制代码](javascript:void(0);)

{

'presets': [

['env', {

'target': {

'chrome': 56

}

}]

]

}

[复制代码](javascript:void(0);)

**Node.js**  
如果通过Babel编译Node.js代码的话，可以设置 "target.node" 是 'current', 含义是 支持的是当前运行版本的nodejs。  
如下配置代码：

[复制代码](javascript:void(0);)

{

"presets": [

["env", {

"targets": {

"node": "current"

}

}]

]

}

[复制代码](javascript:void(0);)

**理解 babel-preset-env 中的选项配置：**  
**1. targets: {[string]: number | string }, 默认为{};**  
含义是支持一个运行环境的对象，比如支持node版本；可以如下配置： node: '6.0';  
运行环境: chrome, opera, edge, firefox, safari, ie, ios, android, node, electron

**2. targets.browsers <Array | string>**  
支持浏览器的配置项，该配置项使用方式可以到 browserslist来查询[（https://github.com/browserslist/browserslist）](https://github.com/browserslist/browserslist" \t "_blank)  
比如上面的 支持每个浏览器最后两个版本和safari大于等于7版本。如上配置。

**3. modules**  
该参数的含义是：启用将ES6模块语法转换为另一种模块类型。将该设置为false就不会转换模块。默认为 'commonjs'.  
该值可以有如下：  
'amd' | 'umd' | 'systemjs' | 'commonjs' | false

我们在项目中一般会看到如下配置，设置modules: false, 如下代码配置：

[复制代码](javascript:void(0);)

"presets": [

'env',

{

'modules': false

}

]

[复制代码](javascript:void(0);)

这样做的目的是：以前我们需要使用babel来将ES6的模块语法转换为AMD, CommonJS，UMD之类的模块化标准语法，但是现在webpack都帮我做了这件事了，所以我们不需要babel来做，因此需要在babel配置项中设置modules为false，因为它默认值是commonjs, 否则的话，会产生冲突。

**4. loose, 该参数值默认为false**。  
含义是：允许它们为这个 preset 的任何插件启用”loose” 转换。

**5. include: 包含一些插件，默认为 [];**  
比如包含箭头函数，可以如下配置：

[复制代码](javascript:void(0);)

{

"presets": [

["env", {

"targets": {

"browsers": ["last 2 versions", "safari >= 7"]

},

"include": ["transform-es2015-arrow-functions", "es6.map"]

}]

]

}

[复制代码](javascript:void(0);)

**6. exclude； 排除哪些插件，默认为 [];**  
比如 排除生成器，可以如下配置：

[复制代码](javascript:void(0);)

{

"presets": [

["env", {

"targets": {

"browsers": ["last 2 versions", "safari >= 7"]

},

"exclude": ["transform-regenerator", "es6.set"]

}]

]

}

[复制代码](javascript:void(0);)

**3.2 理解 babel-presets-stage-x**  
官方预设(preset), 有两种，一个是按年份(babel-preset-es2017)，一个是按阶段(babel-preset-stage-0)。 这主要是根据TC39 委员会ECMASCRPIT 发布流程来制定的。因此到目前为止 有4个不同的阶段预设：

[复制代码](javascript:void(0);)

babel-preset-stage-0

babel-preset-stage-1

babel-preset-stage-2

babel-preset-stage-3

[复制代码](javascript:void(0);)

以上每种预设都依赖于紧随的后期阶段预设，数字越小，阶段越靠后，存在依赖关系。也就是说stage-0是包括stage-1的，以此类推。因此 stage-0包含stage-1/2/3的内容。所以如果我们不知道需要哪个stage-x的话，直接引入stage-0就好了。

**stage0** [(https://babeljs.io/docs/en/babel-preset-stage-0)](https://babeljs.io/docs/en/babel-preset-stage-0)只是一个美好激进的想法，一些 Babel 插件实现了对这些特性的支持 ，但是不确定是否会被定为标准.

**stage1** [(https://babeljs.io/docs/en/babel-preset-stage-1)](https://babeljs.io/docs/en/babel-preset-stage-1) 值得被纳入标准的特性.

**stage2** [(https://babeljs.io/docs/en/babel-preset-stage-2)](https://babeljs.io/docs/en/babel-preset-stage-2) 该特性规范己经被起草，将会被纳入标准里.

**stage3** [(https://babeljs.io/docs/en/babel-preset-stage-3)](https://babeljs.io/docs/en/babel-preset-stage-3)该特性规范已经定稿，大浏览器厂商和 Node.js 社区己开始着手实现.

但是在我们使用的时候只需要安装你想要的阶段就可以了：比如 babel-preset-stage-2， 安装命令如下：

npm install --save-dev babel-preset-stage-2

[回到顶部](https://www.cnblogs.com/tugenhua0707/p/9452471.html#_labelTop)

**二：在webpack中配置babel**

在上面了解了babel后，现在我们需要知道如何在webpack中使用它了。由于babel所做的事情是转换代码，所有需要使用loader去转换，因此我们需要配置babel-loader。

在安装babel-loader之前，我们需要安装babel-core， 因为babel-core是Babel编译器的核心，因此也就意味着如果我们需要使用babel-loader进行es6转码的话，我们首先需要安装 babel-core, 安装命令如下即可：

npm install --save-dev babel-core

然后我们再安装 babel-loader, 命令如下：

npm install --save-dev babel-loader

接着我们需要安装 babel-preset-env, babel-plugin-transform-runtime, babel-preset-stage-2, 如下命令安装

npm install --save-dev babel-preset-env babel-plugin-transform-runtime babel-preset-stage-2

因此 .babelrc 配置如下即可：

[复制代码](javascript:void(0);)

{

"plugins": [

[

"transform-runtime",

{

"polyfill": false

}

]

],

"presets": [

[

"env",

{

"modules": false

}

],

"stage-2"

]

}

[复制代码](javascript:void(0);)

在做demo之前，我们还是先看下目录结构变成如下：

[复制代码](javascript:void(0);)

### 目录结构如下：

demo1 # 工程名

| |--- dist # 打包后生成的目录文件

| |--- node\_modules # 所有的依赖包

| |--- js # 存放所有js文件

| | |-- demo1.js

| | |-- main.js # js入口文件

| |

| |--- webpack.config.js # webpack配置文件

| |--- index.html # html文件

| |--- styles # 存放所有的css样式文件

| |--- .gitignore

| |--- README.md

| |--- package.json

| |--- .babelrc # babel转码文件

[复制代码](javascript:void(0);)

因此webpack配置中需要添加 babel-loader 配置，如下配置：

[复制代码](javascript:void(0);)

module.exports = {

module: {

rules: [

{

test: /\.js$/,

exclude: /(node\_modules)/, // 排除文件

loader: 'babel-loader'

}

]

}

}

[复制代码](javascript:void(0);)

webpack 所有配置如下代码

[复制代码](javascript:void(0);)

const path = require('path');

// 提取css的插件

const ExtractTextPlugin = require('extract-text-webpack-plugin');

const ClearWebpackPlugin = require('clean-webpack-plugin');

module.exports = {

entry: './js/main.js',

output: {

filename: 'bundle.js',

// 将输出的文件都放在dist目录下

path: path.resolve(\_\_dirname, 'dist'),

publicPath: '/dist'

},

mode: 'development',

module: {

rules: [

{

// 使用正则去匹配要用该loader转换的css文件

test: /\.css$/,

loaders: ExtractTextPlugin.extract({

// 转换 .css文件需要使用的Loader

use: ['css-loader']

})

},

{

test: /\.(png|jpg)$/,

loader: 'url-loader',

options: {

limit: 10000,

name: '[name].[ext]'

}

},

{

test: /\.js$/,

exclude: /(node\_modules)/, // 排除文件

loader: 'babel-loader'

}

]

},

resolve: {

// modules: ['plugin', 'js']

},

externals: {

jquery: 'jQuery'

},

devtool: 'source-map',

plugins: [

// new ClearWebpackPlugin(['dist']),

new ExtractTextPlugin({

// 从js文件中提取出来的 .css文件的名称

filename: `main.css`

})

]

};

[复制代码](javascript:void(0);)

package.json 安装依赖包如下：

[复制代码](javascript:void(0);)

{

"name": "demo1",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"dev": "webpack-dev-server --progress --colors --devtool source-map --hot --inline",

"build": "webpack --progress --colors"

},

"devDependencies": {

"babel-core": "^6.26.3",

"babel-loader": "^7.1.5",

"babel-plugin-transform-runtime": "^6.23.0",

"babel-preset-env": "^1.7.0",

"babel-preset-stage-2": "^6.24.1",

"clean-webpack-plugin": "^0.1.19",

"css-loader": "^1.0.0",

"extract-text-webpack-plugin": "^4.0.0-beta.0",

"file-loader": "^1.1.11",

"path": "^0.12.7",

"style-loader": "^0.21.0",

"uglifyjs-webpack-plugin": "^1.2.7",

"url-loader": "^1.0.1",

"webpack": "^4.16.1",

"webpack-cli": "^3.0.8",

"webpack-dev-server": "^3.1.4"

},

"dependencies": {

"axios": "^0.18.0",

"jquery": "^3.3.1"

}

}

[复制代码](javascript:void(0);)

现在我们继续在 main.js 代码内 编写 Generator 函数，代码如下：

[复制代码](javascript:void(0);)

function\* g() {

yield 'a';

yield 'b';

yield 'c';

return 'ending';

}

var gen = g();

console.log(gen.next()); // 返回Object {value: "a", done: false}

for(let a of [1,2,3,4]) {

console.log(a); // 打印出 1, 2, 3, 4

}

[复制代码](javascript:void(0);)

然后重新运行打包命令 npm run dev 后，打开浏览器运行 可以看到控制台输出 {value: "a", done: false}，说明babel已经转译了。