

Step04: 优化 Webpack 配置

在 Step02 中，开发和生产环境相关的配置都集成在 webpack.config.js 一个文件中。为了更好的维护代码，在 Step03 中做一些优化。把 webpack.config.js 拆分成三个部分：

公共配置：把开发和生产环境需要的配置都集中到公共配置文件中，即 webpack.common.js

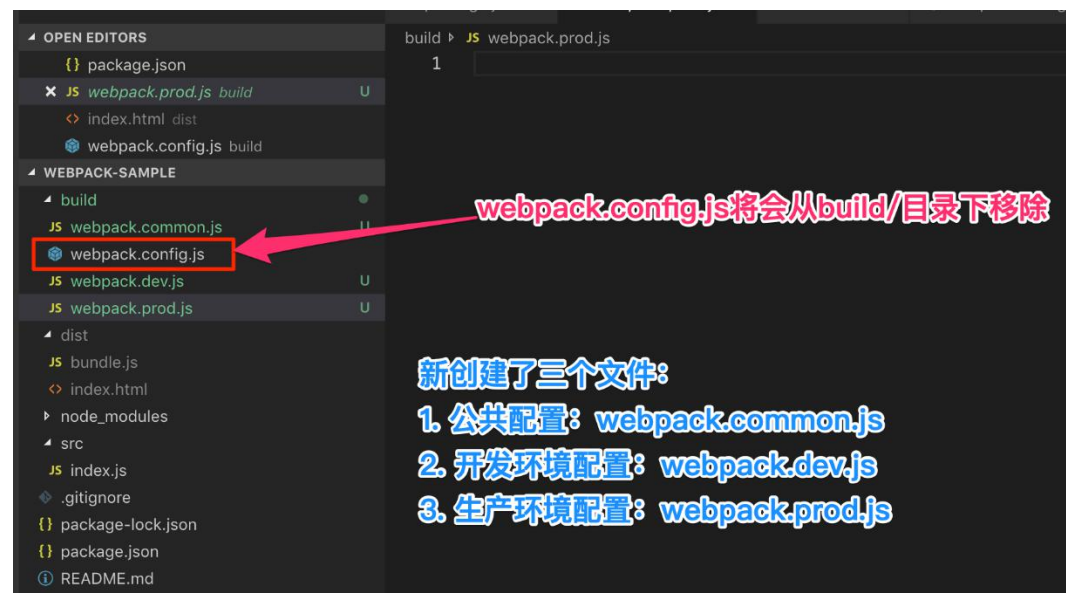
开发环境配置：把开发环境需要的相关配置放置到 webpack.dev.js

生产环境配置：把生产环境需要的相关配置放置到 webpack.prod.js

先在/build 目录下创建上面提到的三个配置文件。在命令终端执行下面的命令即可：

```
⇒ cd build && touch webpack.common.js webpack.dev.js webpack.prod.js
```

这个时候，整个项目目录结构变成下图这样：



遗留下来的 webpack.config.js 文件将会从/build 目录中移除。

Object.assign

为了更好的管理和维护这三个文件，需要安装一个 **webpack-merge** 插件：

yarn add webpack-merge --dev 或者 npm i webpack-merge -D

执行完上面的命令之后，package.json 文件中的 devDependencies 会增加

webpack-merge 相关的配置：

```
// package.json
{
  //... 省略的信息请查看原文件
  "devDependencies": {
    "webpack": "^4.35.0",
    "webpack-cli": "^3.3.5",
    "webpack-dev-server": "^3.7.2",
    "webpack-merge": "^4.2.1"
  }
}
```

接下来分别给 webpack.common.js、webpack.dev.js 和 webpack.prod.js 文件添加相关的配置：

Webpack 公共配置

在公共配置文件 webpack.common.js 文件中添加相应的配置：

```
const webpack = require('webpack');
const path = require('path');
const DIST_PATH = path.resolve(__dirname, '../dist/');
// 声明/dist 的路径
module.exports = {
  // 入口 JS 路径
  // 指示 Webpack 应该使用哪个模块，来作为构建其内部依赖图的开始
  entry: path.resolve(__dirname, '../src/index.js'),
  // 编译输出的 JS 入路径
  // 告诉 Webpack 在哪里输出它所创建的 bundle，以及如何命名这些文件
  output: {
    path: DIST_PATH,
```

```

    // 创建的 bundle 生成到哪里
    filename: 'bundle.js',
    // 创建的 bundle 的名称
  },
  // 模块解析
  module: {},
  // 插件
  plugins: []
}

```

Webpack 开发环境配置

接着给 Webpack 开发环境配置文件 webpack.dev.js 添加下面的相关配置：

```

const webpack = require('webpack');
const path = require('path');
const merge = require('webpack-merge');
const commonConfig = require('./webpack.common.js');
const DIST_PATH = path.resolve(__dirname, '../dist/');

module.exports = merge(commonConfig, {
  mode: 'development',
  // 设置 Webpack 的 mode 模式
  // 开发环境下需要的相关插件配置
  plugins: [],
  // 开发服务器
  devServer: {
    hot: true, // 热更新，无需手动刷新
    contentBase: DIST_PATH,
    // host: '0.0.0.0', // host 地址
    port: 8080, // 服务器端口
    historyApiFallback: true, // 该选项的作用所用 404 都连接到 index.html
    proxy: {
      "/api": "http://localhost:3000" // 代理到后端的服务地址, 会拦截所有以 api 开头的请求地址
    }
  }
})

```

Webpack 生产环境配置

继续给 Webpack 生产环境配置文件 webpack.prod.js 添加相关配置：

```
const webpack = require('webpack');
const path = require('path');
const merge = require('webpack-merge');
const commonConfig = require('./webpack.common.js');
module.exports = merge(commonConfig, {
  mode: 'production',
  // 设置 Webpack 的 mode 模式
  // 生产环境下需要的相关插件配置
  plugins: [ ],
})
```

上面的配置信息只是将 Step02 中 webpack.config.js 分成三个文件来配置，随着后续添加

相应的配置信息，那么这三个文件中的配置信息会越来越多，

修改完 Webpack 的配置之后，对应的 package.json 中的 scripts 中的信息也要做相应的

调整：

```
// package.json
{
  // ... 其他配置信息请查看原文件
  "scripts": {
    "build": "webpack --config ./build/webpack.prod.js --mode production",
    "dev": "webpack-dev-server --config ./build/webpack.dev.js --mode development --open",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
}
```

这个时候重新在命令终端执行

```
// 执行 build 命令，重新打包
⇒ npm run build
// 执行 dev 命令
⇒ npm run dev
```

这仅仅是最基础部分的优化，因为我们的配置还是最简单的，后续我们添加了别的配置之后，

也会在相应的步骤做相应的优化。

Step05:编译 Webpack 项目中的 html 类型的文件

安装: yarn add html-webpack-plugin --dev

首先在项目根目录新建 index.html,作为模板文件进行打包

在插件里配置:

```
new HtmlWebpackPlugin({
  filename: path.resolve(DIST_PATH, 'index.html'), //打包后的文件名
  title: '树鱼虚拟充值生态服务平台', //打包后的页面 title
  template: path.resolve(__dirname, './index.html'), //打包的模板文件
  inject: true,
  hash: true,
  favicon: path.resolve(__dirname, './fav.ico')
})
```

打包后 title 如何自动的添加到页面中呢?

只需要在模板文件的 title 中添加

<title><%= htmlWebpackPlugin.options.title %></title>

参考链接: <https://www.jianshu.com/p/08a60756ffda>

```
var HtmlWebpackPlugin = require('html-webpack-plugin');
```

```
new HtmlWebpackPlugin({
  filename: , //就是 html 文件的文件名, 默认是 index.html
  title: '标题',
  template: , //指定你生成的文件所依赖哪一个 html 文件模板, 模板类型可以是 html、jade、ejs 等
  inject: true, //true body head false
    true 默认值, script 标签位于 html 文件的 body 底部
    body script 标签位于 html 文件的 body 底部
    head script 标签位于 html 文件的 head 中
    false 不插入生成的 js 文件, 这个几乎不会用到的
  hash: true,
  favicon: path.resolve(__dirname, './fav.ico')
  //给你生成的 html 文件生成一个 favicon ,值是一个路径
  //然后再生成的 html 中就有了一个 link 标签
  <link rel="shortcut icon" href="example.ico">
```

```

    minify: //是否对 HTML 进行压缩
    cache: true //默认是 true 的，表示内容变化的时候生成一个新的文件
    showErrors: 当 webpack 报错的时候，会把错误信息包裹再一个 pre 中，默认是 true
    chunks: chunks, //chunks 主要用于多入口文件，当你有多个入口文件，那就回编译后生成多个打包后的文件，那么
chunks 就能选择你要使用那些 js 文件，
    excludeChunks: //排除掉一些 js
  })

```

```

var pluginsAll2 = [];
var pages = glob.sync(path.join(htmlPagesPath, '**/*.html'));
pages.forEach(function(page){
  var pagestr = page.match(/pages\/(\S*)\.html/);
  var name = pagestr[1];
  var plug = new htmlWebpackPlugin({
    filename: path.resolve(buildPath, name + '.html'),
    title: '测试',
    template: path.resolve(htmlPagesPath, name + '.html'),
    inject: true,
    chunks: [name],
    favicon: path.resolve(__dirname, './fav.ico')
  })
  pluginsAll2.push(plug);
})

```

Step06: loaders

<https://www.webpackjs.com/loaders/>

常见的 loader 有：

file-loader, url-loader, style-loader, css-loader, less-loader, sass-loader, babel-loader, raw-loader, vue-loader

webpack >= v2.0.0 开始默认支持 json 文件的导入

webpack 是模块打包工具，而模块不仅是 js，还可以是 css，图片或者其他格式
但是 webpack 默认只知道如何处理 js 模块，那么其他格式的模块处理，和处理方式就需要 loader

如何在 webpack 中配置第三方 loader

配置第三方 loader，需要在 webpack 的配置文件中新增一个 module 节点，节点中是一个一个的规则集合，集合名字是 rules，需要添加 loader 就在 rules 的集合中新增一个规则；每个规则必须的两个配置：

- (1) test : test 搭配的是键值对，值是一个正则表达式，用来匹配要处理的文件类型；
- (2) use : 用来指定使用哪个 loader 模块来打包处理该文件；

```
module:{
  rules:[
    {
      test: /\.xxx$/,
      use:{
        loader: 'xxx-load'
      }
    }
  ]
}
```

当 webpack 处理到不认识的模块时，需要在 webpack 中的 module 处进行配置，当检测到是什什么格式的 模块，使用什什么 loader 来处理

● loader: file-loader: 处理静态资源模块

- 1、作用: file-loader 可以用来帮助 webpack 打包处理一系列的图片文件；比如: .png 、 .jpg 、 .jpeg 等格式的图片；
- 2、使用 file-loader 打包图片的结果: 使用 file-loader 打包的图片会给每张图片都生成一个随机的 hash 值作为图片的名字；

loader: file-loader 原理是把打包入口口中识别出的资源模块，移动到输出目录，并且返回一个地址名称

所以我们什什么时候用 file-loader 呢？ 场景: 就是当我们需要模块，仅仅是从源代码挪移到打包目录，就可以使用 file-loader 来处理， txt, svg, csv, excel, 图片资源啦等等

yarn add file-loader -D 或者 npm install file-loader -D

案例:

```
module: {
  rules: [
```

```

{
  test: /\. (png|jpe?g|gif)$/ ,
  //use 使用用——一个 loader 可以用用对象，字符串串，两个 loader 需要用用数组
  use: {
    loader: "file-loader",
    // options 额外的配置，比比如资源名称
    options: {
      // placeholder 占位符 [name]老老老资源模块的名称
      // [ext]老老资源模块的后缀
      // https://webpack.js.org/loaders/file-loader#placeholders
      name: "[name]_[hash].[ext]",
      //打包后的存放位置
      outputPath: "images/"
    }
  }
}
}
]
},

```

● url-loader :处理图片 base64

当图片较小的时候会把图片转换成 base64 编码，大于 limit 参数的时候还是使用 file-loader 进行拷贝。这样做是好处是可以直接将图片打包到 bundle.js 里，不用额外请求图片（省去 HTTP 请求）

作用： 可以弥补处理file-loader 不能生成 base64 图片格式的缺陷，对小体积的图片比较合适，大图片片不合适。但是，也是需要安装 file-loader

安装：

yarn add url-loader -D 或者 npm install url-loader -D

案例：

```

module: {
  rules: [
    {
      test: /\. (png|jpe?g|gif)$/ ,    //是匹配图片文件后缀名称
      use: {
        loader: "url-loader",    //是指定使用的 loader 和 loader 的配置参数
        options: {
          name: "[name]_[hash].[ext]",
          outputPath: "images/",    //图片打包到的文件目录
          //小于 2048B，才转换成 base64 的文件打成 Base64 的格式，写入 JS
          limit: 2048 ,
          publicPath: '/img'    //最终生成的 CSS 代码中，图片 URL 前缀
        }
      }
    }
  ]
}

```



```

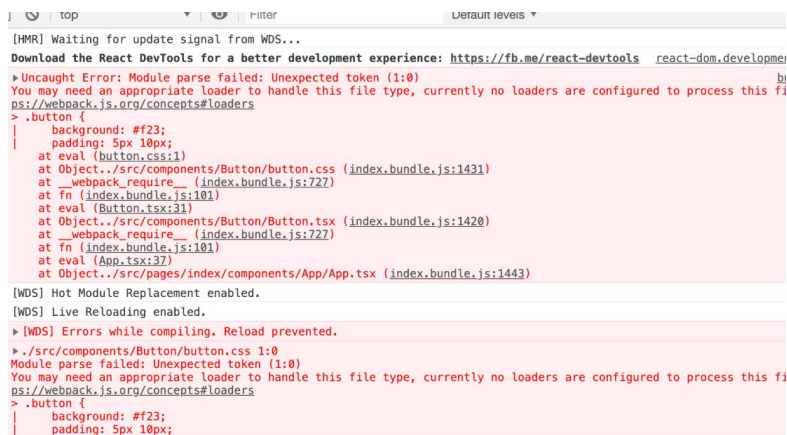
    }
  }
}
],
},

```

● CSS 预处理

(1) CSS loader 配置

如果没有安装 style-loader css-loader 直接引入 css 文件就会报错如下：



```

[HMR] Waiting for update signal from WDS...
Download the React DevTools for a better development experience: https://fb.me/react-devtools
react-dom.development.js:1431
> Uncaught Error: Module parse failed: Unexpected token (1:0)
You may need an appropriate loader to handle this file type, currently no loaders are configured to process this file. See https://webpack.js.org/concepts#loaders
> .button {
  |   background: #f23;
  |   padding: 5px 10px;
  |   at eval (button.css:1)
  |   at Object../src/components/Button/button.css (index.bundle.js:1431)
  |   at __webpack_require__ (index.bundle.js:727)
  |   at fn (index.bundle.js:101)
  |   at eval (Button.tsx:31)
  |   at Object../src/components/Button/Button.tsx (index.bundle.js:1420)
  |   at __webpack_require__ (index.bundle.js:727)
  |   at fn (index.bundle.js:101)
  |   at eval (App.tsx:37)
  |   at Object../src/pages/index/components/App/App.tsx (index.bundle.js:1443)
[WDS] Hot Module Replacement enabled.
[WDS] Live Reloading enabled.
> [WDS] Errors while compiling. Reload prevented.
> ./src/components/Button/button.css 1:0
Module parse failed: Unexpected token (1:0)
You may need an appropriate loader to handle this file type, currently no loaders are configured to process this file. See https://webpack.js.org/concepts#loaders
> .button {
  |   background: #f23;
  |   padding: 5px 10px;
  |

```

这是因为在 Webpack 中没有 CSS 相关的 Loader 配置。那么接下来，来解决

CSS Loader 在 Webpack 中的配置：

- **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
- 而 **vue-style-loader** 是 vue 官方基于 **style-loader** 开发的适用于 vue 的样式解析，
sass-loader 用来解析 sass/scss 文件

两者结合在一起能够把样式嵌入 Webpack 打包后的 JavaScript 文件中

安装: yarn add style-loader css-loader --dev

在

```
module:{
  rules:[]
}
```

配置:

```
{
  test: /\.css$/,
  exclude: /node_modules/,
  use: [{ loader: "style-loader"}, { loader: "css-loader" }] //方式一
//use: ["style-loader", loader: "css-loader" ] //方式二
}
```

(2) 配置 less 环境需要安装:

安装: yarn add less less-loader --dev

配置:

```
{
  test: /\.less$/,
  use: [{ loader: "style-loader" }, { loader: "css-loader" }, { loader: "less-loader" }] //方式一
  use: ['style-loader','css-loader','less-loader'] //方式二
}
```

(3) 配置 scss 环境需要安装:

sass-loader 把 sass 语法转换成 css , 依赖 node-sass 模块

安装: yarn add node-sass sass-loader --dev

//注意: 在项目中要 less 预处理和 sass 预处理二者选其一, 不要同时混着用

loader 有顺序, 从右到左, 从下到上

配置:

```
{
  test: /\.scss$/,
  use: [{ loader: "style-loader" }, { loader: "css-loader" }, { loader: "sass-loader" }] //方式一
}
```

```
use: ['style-loader', 'css-loader', 'sass-loader'] //方式二
}
```

(4) 添加 PostCSS 相关配置

PostCSS 是一个很优秀的东西，他有很多优秀的插件，比如 postcss-preset-env、Autoprefixer 等。另外自己还可以根据自己需要扩展自己想要的 PostCSS 插件。而且在一些移动端的布局中的方案都会需要处理 CSS 单位之间的转换，比如 Flexible (px2rem)，vw-layout (px2vw) 等。

可以说，现代 Web 开发中 PostCSS 相关的配置是工程体系中必不可少的。接下来，我们就看看如何在该工程体系中添加 PostCSS 相关的配置。要配置 PostCSS 相关的事项，需要：

- 安装 postcss、postcss-loader
- 在 Webpack 配置中添加 PostCSS 相关的配置
- 安装自己需要的 PostCSS 插件
- 在 postcss.config.js 或 postcssrc.js 添加有关于 PostCSS 的插件配置

先来执行第一步，安装 postcss 和 postcss-loader：

安装：yarn add postcss-loader postcss --dev

配置：

```
{
  test: /\.css$/,
  exclude: /node_modules/,
  use: [{ loader: "style-loader" }, { loader: "css-loader" }, { loader: 'postcss-loader' } ]
}
```

如果直接启动会报错：No PostCSS Config found....

接下来需要给项目添加 PostCSS 配置相关的信息。先在项目的根目录下创建

postcss.config.js 或 postcssrc.js 文件，并添加相关的配置：

```
// postcss.config.js
module.exports = {
  plugins: {},
}
```

上面的配置还没有添加任何 PostCSS 相关的插件。为了验证我们的配置是否成功，来安装

两个 PostCSS 的插件，

- 添加 css3 前缀

postcss-preset-env 将现代 CSS 转换成浏览器能理解的东西

安装 postcss-preset-env，无需再安装 autoprefixer，由于 postcss-preset-env 已经内置了相关功能。

安装：yarn add postcss-preset-env --dev

并把这两插件的相关配置中添加到 postcss.config.js 中：

```
module.exports = {
  plugins: {
    'postcss-preset-env': {
      autoprefixer: { flexbox: 'no-2009' }, //可去掉
      stage: 3
    },
  }
}
```

如果安装了自动补全插件 autoprefixer， yarn add autoprefixer --dev 也可以通过这个配置来生效

```
module.exports = {
  plugins: {
    'autoprefixer': {}
  }
}
```

在样式文件中添加一些样式代码来验证是否生效了：

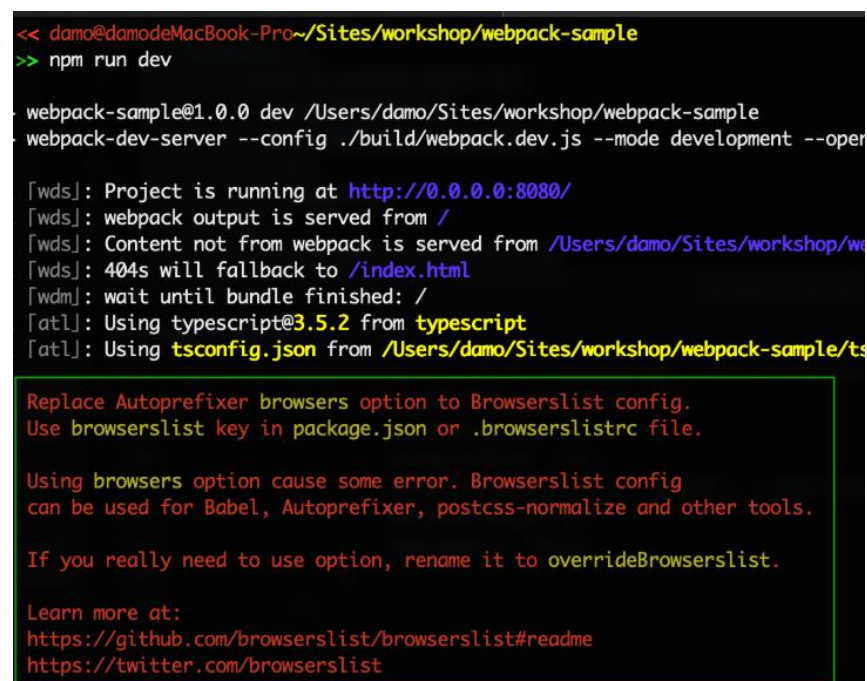
```
.box {
  width: 50px;
```

```
height: 30px;
background: url(../images/ABC.jpg);
transform: rotate(30deg);
}
```

如果生效，会是这样的

```
.box
{
transform: rotate(7deg);
-ms-transform: rotate(7deg);      /* IE 9 */
-moz-transform: rotate(7deg);     /* Firefox */
-webkit-transform: rotate(7deg); /* Safari 和 Chrome */
-o-transform: rotate(7deg);      /* Opera */
}
```

如果就这样执行 `npm run dev`，会有 autoprefixer 相关的警告信息：



```
<< damo@damodeMacBook-Pro~/Sites/workshop/webpack-sample
>> npm run dev

webpack-sample@1.0.0 dev /Users/damo/Sites/workshop/webpack-sample
webpack-dev-server --config ./build/webpack.dev.js --mode development --open

[wds]: Project is running at http://0.0.0.0:8080/
[wds]: webpack output is served from /
[wds]: Content not from webpack is served from /Users/damo/Sites/workshop/w
[wds]: 404s will fallback to /index.html
[wdm]: wait until bundle finished: /
[atl]: Using typescript@3.5.2 from typescript
[atl]: Using tsconfig.json from /Users/damo/Sites/workshop/webpack-sample/ts

Replace Autoprefixer browsers option to Browserslist config.
Use browserslist key in package.json or .browserslistrc file.

Using browsers option cause some error. Browserslist config
can be used for Babel, Autoprefixer, postcss-normalize and other tools.

If you really need to use option, rename it to overrideBrowserslist.

Learn more at:
https://github.com/browserslist/browserslist#readme
https://twitter.com/browserslist
```

主要是因为 `postcss-preset-env` 插件涵盖了 autoprefixer 插件的能力，要解决这个警告信

息，需要在 `package.json` 中添加 browserslist 相关的配置信息：

```
// package.json
{
  // ... 省略的信息可以查看原文件
  "browserslist": [
    "last 5 version",
    "> 2%",
```

```
"IE 8"  
],  
}
```

● 单位转换 px - rem

把项目中 px 单位转化为 rem,实现响应式,

yarn add postcss-pxtorem -D 或者 npm install postcss-pxtorem -D

```
module.exports = {  
  plugins: {  
    'postcss-preset-env': {  
      autoprefixer: { flexbox: 'no-2009' },  
      stage: 3  
    },  
    'postcss-pxtorem': {  
      rootValue: 10,  
      minPixelValue: 2,  
      propWhitelist: []  
    }  
  }  
}
```

(5) css 抽离 后面插件讲

配置:

```
{  
  test: /\.css$/,  
  use: ExtractTextPlugin.extract({  
    fallback: 'style-loader',  
    use: [{  
      loader: 'css-loader',  
      options: { importLoaders: 1 }  
    },  
    { loader: 'postcss-loader',  
      options: { //方式一  
        plugins: [  
          require("autoprefixer")  
        ]  
      }  
    }  
  ]  
}
```

```
    },  
    {loader:'less-loader'}  
  ]),  
}
```

方式二:

在项目根目录 添加一个 postcss.config.js

```
module.exports = {  
  plugins:[  
    require("autoprefixer")  
  ]  
}
```

在 package.json 添加

```
"browserslist": [  
  "last 1 version",  
  "> 1%",  
  "IE 10"  
],
```

Step07:增加 babel支持

babel7 的一些变化

preset 的变更:

淘汰 es201x, 删除 stage-x, 推荐 env

如果你还在使用 es201x, 官方建议使用 env 进行替换。淘汰并不是删除, 只是不推荐使用。

但 stage-x 是直接被删了, 也就是说在 babel7 中使用 es201X 是会报错的。

包名称变化

babel 7 的一个重大变化, 把所有 babel-* 重命名为 @babel/*,

例如:

babel-cli —> @babel/cli。

babel-preset-env —> @babel/preset-env

安装: yarn add @babel/core babel-loader @babel/preset-env -D

其中 preset 是 babel 的配置, babel-loader 在 webpack 中解析 babel 中会用到, polyfill 也是 babel 的一个依赖在 webpack 中解析 babel 中会用到

//babel-loader 是 webpack 与 babel 的通信桥梁, 在 webpack 中解析 babel 中会用到, 不会做把 es6 转成 es5 的工作, 这部分工作需要用到 @babel/preset-env 来做

//@babel/preset-env 里包含了 es6 转 es5 的转换规则

babel-preset-es2015 已经被 babel-preset-env 替代, 现在已经是 @babel/preset-env

@babel/preset-env 这个比较强大, 是一个面向未来得库, 即使是 es7,8,9 出来以后, 都可以帮忙处理

在开发依赖包安装如下插件:

babel-loader

@babel/core

@babel/preset-env

@babel/plugin-transform-runtime

@babel/plugin-transform-modules-commonjs

在线上依赖包安装

@babel/runtime

<https://babeljs.io/setup#installation>

//babel 配置

```
{
  test: /\.jsx?$/,
```



```

use:{
  loader:'babel-loader',
  options:{
    presets:[ "@babel/preset-env" ] //方法一
  }
},
exclude:/node_modules/
}

```

.babelrc 配置： 方式二

```

{
  "presets":["@babel/preset-env"]
}

```

然后可以把配置文件里的配置 babel 的地方去掉 presets 配置

测试代码：

```

//index.js
const arr = [new Promise(() => {}), new Promise(() => {})];
arr.map(item => { console.log(item); });

```

```

var isCol = ['11','22'].includes('11');
console.log(isCol);

```

在 ie10 会报错

```

+++
❌ SCRIPT438: 对象不支持“includes”属性或方法
index.js (33,1)

```

通过上面的几步 还不够，includes Promise 等一些还没有转换过来，这时候需要借助@babel/polyfill，把 es 的新特性都装进来，来弥补低版本浏览器中缺失的特性

babel-polyfill

以全局变量的方式注入进来的。window.Promise，它会造成全局对象的污染

安装： yarn add @babel/polyfill --dev

Webpack.config.js 配置：

```

{
  test: /\.js$/,

```

```

exclude: /node_modules/,
loader: "babel-loader",
options: {
  presets: ["@babel/preset-env"]
}}

```

//index.js 顶部

```
import "@babel/polyfill";
```

会发现打包的体积大了很多，这是因为 **polyfill** 默认会把所有特性注入进来，假如我想我用到的 **es6+**，才会注入，没用到的不注入，从而减少打包的体积，可不可以呢

当然可以

修改 Webpack.config.js

```

options: {
presets: [
  [
"@babel/preset-env",
{
  targets: {
    edge: "17",
    firefox: "60",
    chrome: "67",
    safari: "11.1"
  },
useBuiltIns: "usage"//按需注入 ,
"corejs": "2", // 声明 corejs 版本 ,注意这是是版本 2

}}
]]
}

```

进一步测试，这个时候 **ie9** 是可以运行，但是 **IE8** 提示



如何解决呢？

查了些资料，发现 **object.defineProperty** 在 **ie8** 中存在，但是只用于操作 **dom** 对象，难怪他会报错对象不支持该操作，安装一个 **object-defineproperty-ie8** 的包 试试，还是不兼容？

当我们开发的是组件库，工具库这些场景的时候，**polyfill** 就不适合了，因为 **polyfill** 是注入到全局变量，**window** 下的，会污染全局环境，所以推荐闭包方式：

```
@babel/plugin-transform-runtime
```

@babel/plugin-transform-runtime

它不会造成全局污染

```
yarn add @babel/plugin-transform-runtime --dev
```

```
yarn add @babel/runtime --save //注意这里安装到 dependencies 的依赖包
```

怎么使用？

先注释掉 index.js 里的 polyfill

修改配置文件：注释掉之前的 presets，添加 plugins

```
options: {
  presets: [
    [
      "@babel/preset-env",
      {
        targets: {
          edge: "17",
          firefox: "60",
          chrome: "67",
          safari: "11.1"
        },
        useBuiltIns: "usage",
        corejs: 2
      }
    ],
    "plugins": [
      [
        "@babel/plugin-transform-runtime",
        {
          "absoluteRuntime": false,
          "corejs": 2,
          "helpers": true,
          "regenerator": true,
          "useESModules": false
        }
      ]
    ]
  ]
}
```

```

ERROR in ./src/index.js
Module not found: Error: Can't resolve '@babel/runtime-corejs2/core-js/promise'
  \src'
  @ ./src/index.js 1:0-62 39:13-21

ERROR in ./src/index.js
Module not found: Error: Can't resolve '@babel/runtime-corejs2/core-js/promise' in 'D:\ruanmou
  \src'
  @ ./src/index.js 5:38-87

ERROR in ./src/index.js
Module not found: Error: Can't resolve '@babel/runtime-corejs2/helpers/interopRequireDefault'
  inline\B-test01\src'
  @ ./src/index.js 3:29-92

```

如果出现这个错误，记得安装 @babel/runtime-corejs2

yarn add @babel/runtime-corejs2

<https://babeljs.io/docs/en/babel-runtime-corejs2>

useBuiltIns 选项是 babel 7 的新功能，这个选项告诉 babel 如何配置 @babel/polyfill。它有三个参数可以使用：

- ①entry: 需要在 webpack 的入口文件里 import "@babel/polyfill" 一次。babel 会根据你的使用情况导入垫片，没有使用的功能不会被导入相应的垫片。
- ②usage: 不需要 import，全自动检测，但是要安装 @babel/polyfill。（试验阶段）
- ③false: 如果你 import "@babel/polyfill"，它不会排除掉没有使用的垫片，程序体积会庞大。（不推荐）

请注意：usage 的行为类似 babel-transform-runtime，不会造成全局污染，因此也不会对类似 Array.prototype.includes() 进行 polyfill。

扩展：

babelrc 文件：新建.babelrc 文件，把 options 部分移入到该文件中，就可以了

//.babelrc

```

{ "plugins": [
  [
    "@babel/plugin-transform-runtime",    {
      "absoluteRuntime": false,
      "corejs": 2,
      "helpers": true,
      "regenerator": true,
      "useESModules": false
    }
  ]
]}

```

//webpack.config.js

```

{
  test: /\.js$/,
  exclude: /node_modules/,

```

```
    loader: "babel-loader"
  }
}
```

方式一：

使用 webpack，有多种方法可以包含 polyfill

- 当与 babel-preset-env 一起使用时,如果在.babelrc 中指定了 useBuiltIns: 'usage',那么在 webpack.config.js 配置的 entry 处或源代码中都不要包含 babel-polyfill。注意，babel-polyfill 仍然需要安装。
- 如果在.babelrc 中指定了 useBuiltIns: 'entry'，那么在应用程序入口点的顶部包括 babel-polyfill，方法是按照上面讨论的 require 或 import。
- 如果在.babelrc 中未指定 useBuiltIns 键，或者使用 useBuiltIns: false 设置了该键，则直接将 babel-polyfill 添加到 webpack.config.js 中的 entry 中。

方案	打包后大小	优点	缺点
babel-polyfill	259K	完整模拟ES2015+环境	体积过大；污染全局对象和内置的对象原型
babel-runtime	63K	按需引入，打包体积小	不模拟实例方法
babel-preset-env (开启 useBuiltIns)	194K	按需引入，可配置性高	-

```
{
  "presets": [
    [
      "@babel/preset-env", {
        "useBuiltIns": "usage", //entry,false
        "targets": {
          "browsers": [ "> 1%", "last 5 versions", "ie >= 8" ]
        }
      }
    ]
  ],
  "plugins": [
    "@babel/plugin-transform-runtime",
    "@babel/plugin-transform-modules-commonjs"
  ]
}
```

在引入@babel/preset-env (一个帮你配置 babel 的 preset, 根据配置的目标环境自动采用需要的 babel 插件) 配置 babel 中的 useBuiltIns: 'usage'时, 编译出现又出现这个警告提示

```
WARNING: We noticed you're using the `useBuiltIns` option without declaring a core-js version. Currently, we assume version 2.x when no version is passed. Since this default version will likely change in future versions of Babel, we recommend explicitly setting the core-js version you are using via the `corejs` option. You should also be sure that the version you pass to the `corejs` option matches the version specified in your `package.json`'s `dependencies` section. If it doesn't, you need to run one of the following commands:

npm install --save core-js@2    npm install --save core-js@3
yarn add core-js@2             yarn add core-js@3
```

问题解释: 警告在使用 useBuiltIns 选项时, 需要声明 core-js 的版本

解决办法:

- 1) 方法一: 安装 yarn add core-js@2
- 2) 方法二: 在 .babelrc 文件中添加 corejs:"2", 申明 corejs 的版本

```
{
  "presets": [
    [
      "@babel/preset-env", {
        "useBuiltIns": "usage",
        "corejs": "2", // 声明 corejs 版本
        "targets": {
          "browsers": [ "> 1%", "last 5 versions", "ie >= 8" ]
        }
      }
    ]
  ],
  "plugins": [
    "@babel/plugin-transform-runtime",
    "@babel/plugin-transform-modules-commonjs"
  ]
}
```

完美解决问题!

目前还不兼容 IE8



项目如果是用 babel7 来转译, 需要安装 @babel/core、@babel/preset-env 和 @babel/plugin-transform-runtime, 而不是 babel-core、babel-preset-env 和 babel-plugin-transform-runtime, 它们是由于 babel6 的

方式二: 下面的 2 种情况都不推荐

```
// 情况一: 在 webpack 中引用
module.exports = {
  entry: ["babel-polyfill", "./app.js"] //不推荐
};
```

```
// 情况二: 在 js 入口顶部引入
import "babel-polyfill"; 不推荐
```

<https://babeljs.io/docs/en/6.26.3/babel-polyfill>