---

link: null
title: 珠峰架构师成长计划
description: src/index.js
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=57 sentences=59, words=690

---

1. babel-polyfill

- 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
    - 在全局对象上的方法,比如说ES6在Array对象上新增了 Array.find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
- Babel 7.4之后不再推荐使用@babel/polyfill
- babel v7 推荐使用@babel/preset-env代替以往的诸多polyfill方案
- babel-preset-env (https://babeljs.io/docs/en/babel-preset-env)
- babel-polyfill (https://babeljs.io/docs/en/babel-polyfill)
- babel-runtime (https://babeljs.io/docs/en/babel-runtime)
- babel-plugin-transform-runtime (https://babeljs.io/docs/en/babel-plugin-transform-runtime)

```
npm i @babel/polyfill
```

```
{
    test: /\.js?$/,
    exclude: /node_modules/,
    use: {
        loader: 'babel-loader',
        options: {
            targets: {

                "browsers": [">1%"]
            }
        }
    }
}
```

- 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/preset-env (https://www.npmjs.com/package/@babel/preset-env)为每一个环境的预设
- @babel/preset-env默认只支持语法转化，需要开启 useBuiltIns配置才能转化API和实例方法
- useBuiltIns可选值包括："usage" | "entry" | false, 默认为 false，表示不对 polyfills 处理，这个配置是引入 polyfills 的关键
- useBuiltIns: false 此时不对 polyfill 做操作。如果引入 @babel/polyfill，则无视配置的浏览器兼容，引入所有的 polyfill

src/index.js

```
import '@babel/polyfill';
let sum = (a, b) => a + b;
let promise = Promise.resolve();
console.log([1, 2, 3].find(item => item === 2));
```

webpack.config.js

```
const path = require('path');
module.exports = {
    mode: 'development',
    devtool: false,
    entry: './src/index.js',
    output: {
        path: path.resolve(__dirname, 'dist'),
        filename: 'main.js'
    },
    module: {
        rules: [
            {
                test: /\.js?$/,
                exclude: /node_modules/,
                use: {
                    loader: 'babel-loader',
                    options: {
                        sourceType: "unambiguous",
+                       presets: [["@babel/preset-env", { useBuiltIns: false }]]
                    }
                }
            }
        ]
    },
    plugins: []
};
```

- 在项目入口引入一次（多次引入会报错）
- "useBuiltIns": "entry" 根据配置的浏览器兼容，引入浏览器不兼容的 polyfill。需要在入口文件手动添加 import '@babel/polyfill'，会自动根据 browserslist 替换成浏览器不兼容的所有 polyfill
- 这里需要指定 core-js 的版本, corejs默认是2,
- 如果配置 corejs: 3, 则 import '@babel/polyfill' 需要改成 import 'core-js/stable';import 'regenerator-runtime/runtime';
-
    - corejs默认是2
- 50.8 KiB

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

src\index.js core-js@2

```
import '@babel/polyfill';
let sum = (a, b) => a + b;
let promise = Promise.resolve();
console.log([1, 2, 3].find(item => item === 2));
```

core-js@3

```
import 'core-js/stable';
import 'regenerator-runtime/runtime';
let sum = (a, b) => a + b;
let promise = Promise.resolve();
console.log([1, 2, 3].find(item => item === 2));
```

```
{
  test: /\.js?$/,
  exclude: /node_modules/,
  use: {
      loader: 'babel-loader',
      options: {
+         presets: [["@babel/preset-env", { useBuiltIns: 'entry', corejs: { version: 3 } }]]
      }
  }
}
```

```
WARNING (@babel/preset-env): 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
```

- "useBuiltIns": "usage" usage 会根据配置的浏览器兼容，以及你代码中用到的 API 来进行 polyfill，实现了按需添加
- 当设置为usage时，polyfill会自动按需添加，不再需要手工引入 @babel/polyfill

src/index.js

```
console.log([1, 2, 3].find(item => item === 2));
console.log(Array.prototype.find);
console.log(Array.prototype.hasOwnProperty('find'));
```

webpack.config.js

```
{
  test: /\.js?$/,
  exclude: /node_modules/,
  use: {
      loader: 'babel-loader',
      options: {
+         presets: [["@babel/preset-env", { useBuiltIns: 'usage', corejs: { version: 3 } }]]
      }
  }
}
```

2. babel-runtime

- Babel为了解决全局空间污染的问题，提供了单独的包babel-runtime (https://babeljs.io/docs/en/babel-runtime)用以提供编译模块的工具函数
- 简单说 babel-runtime 更像是一种按需加载的实现，比如你哪里需要使用 Promise，只要在这个文件头部 import Promise from 'babel-runtime/core-js/promise'就行了

```
npm i babel-runtime --save-dev
```

src/index.js

```
import Promise from 'babel-runtime/core-js/promise';
const p = new Promise(()=> {});
```

3. babel-plugin-transform-runtime

- @babel/plugin-transform-runtime插件是为了解决
    - 多个文件重复引用 相同helpers(帮助函数)=>提取运行时
    - 新API方法全局污染 -> 局部引入
- 启用插件 babel-plugin-transform-runtime后，Babel就会使用 babel-runtime下的工具函数
- babel-plugin-transform-runtime插件能够将这些工具函数的代码转换成 require语句，指向为对 babel-runtime的引用
- babel-plugin-transform-runtime 就是可以在我们使用新 API 时自动 import babel-runtime 里面的 polyfill
    - 当我们使用 async/await 时，自动引入 babel-runtime/regenerator
    - 当我们使用 ES6 的静态事件或内置对象时，自动引入 babel-runtime/core-js
    - 移除内联 babel helpers并替换使用 babel-runtime/helpers 来替换

```
npm i @babel/plugin-transform-runtime @babel/runtime-corejs3 --save-dev
```

webpack.config.js

```
{
   test: /\.js?$/,
   exclude: /node_modules/,
   use: {
       loader: 'babel-loader',
       options: {
           sourceType: "unambiguous",
           presets: [["@babel/preset-env", { useBuiltIns: 'usage', corejs: { version: 3 } }]],
           plugins: [
               [
                   "@babel/plugin-transform-runtime",
                   {
                       corejs: 3,
                       helpers: true,
                       regenerator: true
                   }
               ],
           ]
       }
   }
}
```

- 当我们使用 ES6 的静态事件或内置对象时自动引入 babel-runtime/core-js

```
const p = new Promise(() => { });
console.log(p);
```

- 移除内联babel helpers并替换使用 `babel-runtime/helpers` 来替换
- 避免内联的 helper 代码在多个文件重复出现

```
class A {

}
class B extends A {

}
console.log(new B());
```

- 是否开启 `generator`函数转换成使用 `regenerator runtime`来避免污染全局域

```

function* gen() {

}
console.log(gen());
```

4. 最佳实践

- @babel/preset-env和plugin-transform-runtime二者都可以设置使用corejs来处理polyfill
- useBuiltIns使用usage
- plugin-transform-runtime只使用其移除内联复用的辅助函数的特性，减小打包体积

```
{
  "presets": [
      [
          "@babel/preset-env",
          {
              "useBuiltIns": "usage",
              "corejs": 3
          }
      ]
  ],
  "plugins": [
      [
          "@babel/plugin-transform-runtime",
          {
              "corejs": false
          }
      ]
  ]
}
```

- 类库开发尽量不使用污染全局环境的 `polyfill`，因此 `@babel/preset-env`只发挥语法转换的功能
- polyfill由 `@babel/plugin-transform-runtime`来处理，推荐使用core-js@3

```
{
    "presets": [
        [
            "@babel/preset-env"
        ]
    ],
    "plugins": [
        [
            "@babel/plugin-transform-runtime",
            {
                "corejs": {
                    "version": 3
                }
            }
        ]
    ]
}
```

5. polyfill-service

- polyfill.io (https://polyfill.io/v3/)自动化的 JavaScript Polyfill 服务
- polyfill.io (https://polyfill.io/v3/)通过分析请求头信息中的 UserAgent 实现自动加载浏览器所需的 polyfills

```

```