## **Babel 转码器**

[Babel](https://babeljs.io/) 是一个广泛使用的 ES6 转码器，可以将 ES6 代码转为 ES5 代码，从而在现有环境执行。这意味着，你可以用 ES6 的方式编写程序，又不用担心现有环境是否支持。下面是一个例子。

*// 转码前*

input.map(item => item + 1);

*// 转码后*

input.map(**function** (item) {

**return** item + 1;

});

上面的原始代码用了箭头函数，Babel 将其转为普通函数，就能在不支持箭头函数的 JavaScript 环境执行了。

### **配置文件**.babelrc

Babel 的配置文件是.babelrc，存放在项目的根目录下。使用 Babel 的第一步，就是配置这个文件。

该文件用来设置转码规则和插件，基本格式如下。

{

"presets": [],

"plugins": []

}

presets字段设定转码规则，官方提供以下的规则集，你可以根据需要安装。

*# 最新转码规则*

$ npm install --save-dev babel-preset-latest

*# react 转码规则*

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

*# 不同阶段语法提案的转码规则（共有4个阶段），选装一个*

$ npm install --save-dev babel-preset-stage-0

$ npm install --save-dev babel-preset-stage-1

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

$ npm install --save-dev babel-preset-stage-3

然后，将这些规则加入.babelrc。

{

"presets": [

"latest",

"react",

"stage-2"

],

"plugins": []

}

注意，以下所有 Babel工具和模块的使用，都必须先写好.babelrc。

### **命令行转码**babel-cli

Babel提供babel-cli工具，用于命令行转码。

它的安装命令如下。

$ npm install --global babel-cli

基本用法如下。

*# 转码结果输出到标准输出*

$ babel example.js

*# 转码结果写入一个文件# --out-file 或 -o 参数指定输出文件*

$ babel example.js --out-file compiled.js*# 或者*

$ babel example.js -o compiled.js

*# 整个目录转码# --out-dir 或 -d 参数指定输出目录*

$ babel src --out-dir lib*# 或者*

$ babel src -d lib

*# -s 参数生成source map文件*

$ babel src -d lib -s

上面代码是在全局环境下，进行 Babel 转码。这意味着，如果项目要运行，全局环境必须有 Babel，也就是说项目产生了对环境的依赖。另一方面，这样做也无法支持不同项目使用不同版本的 Babel。

一个解决办法是将babel-cli安装在项目之中。

*# 安装*

$ npm install --save-dev babel-cli

然后，改写package.json。

{

*// ...*

"devDependencies": {

"babel-cli": "^6.0.0"

},

"scripts": {

"build": "babel src -d lib"

},

}

转码的时候，就执行下面的命令。

$ npm run build

### **babel-node**

babel-cli工具自带一个babel-node命令，提供一个支持ES6的REPL环境。它支持Node的REPL环境的所有功能，而且可以直接运行ES6代码。

它不用单独安装，而是随babel-cli一起安装。然后，执行babel-node就进入REPL环境。

$ babel-node

> (x => x \* 2)(1)

2

babel-node命令可以直接运行ES6脚本。将上面的代码放入脚本文件es6.js，然后直接运行。

$ babel-node es6.js

2

babel-node也可以安装在项目中。

$ npm install --save-dev babel-cli

然后，改写package.json。

{

"scripts": {

"script-name": "babel-node script.js"

}

}

上面代码中，使用babel-node替代node，这样script.js本身就不用做任何转码处理。

### **babel-register**

babel-register模块改写require命令，为它加上一个钩子。此后，每当使用require加载.js、.jsx、.es和.es6后缀名的文件，就会先用Babel进行转码。

$ npm install --save-dev babel-register

使用时，必须首先加载babel-register。

require("babel-register");

require("./index.js");

然后，就不需要手动对index.js转码了。

需要注意的是，babel-register只会对require命令加载的文件转码，而不会对当前文件转码。另外，由于它是实时转码，所以只适合在开发环境使用。

### **babel-core**

如果某些代码需要调用 Babel 的 API 进行转码，就要使用babel-core模块。

安装命令如下。

$ npm install babel-core --save

然后，在项目中就可以调用babel-core。

**var** babel = require('babel-core');

*// 字符串转码*

babel.transform('code();', options);*// => { code, map, ast }*

*// 文件转码（异步）*

babel.transformFile('filename.js', options, **function**(err, result) {

result; *// => { code, map, ast }*

});

*// 文件转码（同步）*

babel.transformFileSync('filename.js', options);*// => { code, map, ast }*

*// Babel AST转码*

babel.transformFromAst(ast, code, options);*// => { code, map, ast }*

配置对象options，可以参看官方文档<http://babeljs.io/docs/usage/options/>。

下面是一个例子。

**var** es6Code = 'let x = n => n + 1';**var** es5Code = require('babel-core')

.transform(es6Code, {

presets: ['latest']

})

.code;*// '"use strict";\n\nvar x = function x(n) {\n return n + 1;\n};'*

上面代码中，transform方法的第一个参数是一个字符串，表示需要被转换的ES6代码，第二个参数是转换的配置对象。

### **babel-polyfill**

Babel 默认只转换新的 JavaScript 句法（syntax），而不转换新的 API，比如Iterator、Generator、Set、Maps、Proxy、Reflect、Symbol、Promise等全局对象，以及一些定义在全局对象上的方法（比如Object.assign）都不会转码。

举例来说，ES6在Array对象上新增了Array.from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，为当前环境提供一个垫片。

安装命令如下。

$ npm install --save babel-polyfill

然后，在脚本头部，加入如下一行代码。

**import** 'babel-polyfill';*// 或者*require('babel-polyfill');

Babel 默认不转码的 API 非常多，详细清单可以查看babel-plugin-transform-runtime模块的[definitions.js](https://github.com/babel/babel/blob/master/packages/babel-plugin-transform-runtime/src/definitions.js)文件。

### **浏览器环境**

Babel 也可以用于浏览器环境。但是，从 Babel 6.0 开始，不再直接提供浏览器版本，而是要用构建工具构建出来。如果你没有或不想使用构建工具，可以使用[babel-standalone](https://github.com/Daniel15/babel-standalone)模块提供的浏览器版本，将其插入网页。

<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-standalone/6.4.4/babel.min.js"></script><script type="text/babel">*// Your ES6 code*</script>

注意，网页实时将 ES6 代码转为 ES5，对性能会有影响。生产环境需要加载已经转码完成的脚本。

下面是如何将代码打包成浏览器可以使用的脚本，以Babel配合Browserify为例。首先，安装babelify模块。

$ npm install --save-dev babelify babel-preset-latest

然后，再用命令行转换 ES6 脚本。

$ browserify script.js -o bundle.js \

-t [ babelify --presets [ latest ] ]

上面代码将ES6脚本script.js，转为bundle.js，浏览器直接加载后者就可以了。

在package.json设置下面的代码，就不用每次命令行都输入参数了。

{

"browserify": {

"transform": [["babelify", { "presets": ["latest"] }]]

}

}

### **在线转换**

Babel 提供一个[REPL在线编译器](https://babeljs.io/repl/)，可以在线将 ES6 代码转为 ES5 代码。转换后的代码，可以直接作为 ES5 代码插入网页运行。

### **与其他工具的配合**

许多工具需要 Babel 进行前置转码，这里举两个例子：ESLint 和 Mocha。

ESLint 用于静态检查代码的语法和风格，安装命令如下。

$ npm install --save-dev eslint babel-eslint

然后，在项目根目录下，新建一个配置文件.eslintrc，在其中加入parser字段。

{

"parser": "babel-eslint",

"rules": {

...

}

}

再在package.json之中，加入相应的scripts脚本。

{

"name": "my-module",

"scripts": {

"lint": "eslint my-files.js"

},

"devDependencies": {

"babel-eslint": "...",

"eslint": "..."

}

}

Mocha 则是一个测试框架，如果需要执行使用 ES6 语法的测试脚本，可以修改package.json的scripts.test。

"scripts": {

"test": "mocha --ui qunit --compilers js:babel-core/register"

}

上面命令中，--compilers参数指定脚本的转码器，规定后缀名为js的文件，都需要使用babel-core/register先转码。