

ECMAScript 6 入门

作者：阮一峰

授权：署名-非商用许可证



目录

- 0.前言
- 1.ECMA Script 6 简介
- 2.let 和 const 命令
- 3.变量的解构赋值
- 4.字符串的扩展
- 5.正则的扩展
- 6.数值的扩展
- 7.函数的扩展
- 8.数组的扩展
- 9.对象的扩展
- 10.对象的新增方法
- 11.Symbol
- 12.Set 和 Map 数据结构
- 13.Proxy
- 14.Reflect
- 15.Promise 对象
- 16.Iterator 和 for...of 循环
- 17.Generator 函数的语法
- 18.Generator 函数的异步应用
- 19.async 函数
- 20.Class 的基本语法
- 21.Class 的继承
- 22.Decorator
- 23.Module 的语法
- 24.Module 的加载实现
- 25.编程风格
- 26.读懂规格
- 27.ArrayBuffer
- 28.最新提案
- 29.参考链接

其他

- 源码
- 修订历史
- 反馈意见

ECMAScript 6 简介

- 1.ECMA Script 和 JavaScript 的关系
- 2.ES6 与 ECMA Script 2015 的关系
- 3.语法提案的批准流程
- 4.ECMA Script 的历史
- 5.部署进度
- 6.Babel 转码器
- 7.Traceur 转码器

ECMAScript 6.0（以下简称 ES6）是 JavaScript 语言的下一代标准，已经在 2015 年 6 月正式发布了。它的目标，是使得 JavaScript 语言可以用来编写复杂的大型应用程序，成为企业级开发语言。

1. ECMAScript 和 JavaScript 的关系

一个常见的问题是，ECMAScript 和 JavaScript 到底是什么关系？

要讲清楚这个问题，需要回顾历史。1996 年 11 月，JavaScript 的创造者 Netscape 公司，决定将 JavaScript 提交给标准化组织 ECMA，希望这种语言能够成为国际标准。次年，ECMA 发布 262 号标准文件（ECMA-262）的第一版，规定了浏览器脚本语言的标准，并将这种语言称为 ECMAScript，这个版本就是 1.0 版。

该标准从一开始就是针对 JavaScript 语言制定的，但是之所以不叫 JavaScript，有两个原因。一是商标，Java 是 Sun 公司的商标，根据授权协议，只有 Netscape 公司可以合法地使用 JavaScript 这个名字，且 JavaScript 本身也已经被 Netscape 公司注册为商标。二是想体现这门语言的制定者是 ECMA，不是 Netscape，这样有利于保证这门语言的开放性和中立性。

因此，ECMAScript 和 JavaScript 的关系是，前者是后者的规格，后者是前者的一种实现（另外的 ECMAScript 方言还有 Jscript 和 ActionScript）。日常场合，这两个词是可以互换的。

2. ES6 与 ECMAScript 2015 的关系

ECMAScript 2015（简称 ES2015）这个词，也是经常可以看到的。它与 ES6 是什么关系呢？

2011 年，ECMAScript 5.1 版发布后，就开始制定 6.0 版了。因此，ES6 这个词的原意，就是指 JavaScript 语言的下一个版本。

但是，因为这个版本引入的语法功能太多，而且制定过程当中，还有很多组织和个人不断提交新功能。事情很快就变得清楚了，不可能在一个版本里面包括所有将要引入的功能。常规的做法是先发布 6.0 版，过一段时间再发 6.1 版，然后是 6.2 版、6.3 版等等。

但是，标准的制定者不想这样做。他们想让标准的升级成为常规流程：任何人在任何时候，都可以向标准委员会提交新语法的提案，然后标准委员会每个月开一次会，评估这些提案是否可以接受，需要哪些改进。如果经过多次会议以后，一个提案足够成熟了，就可以正式进入标准了。这就是说，标准的版本升级成为了一个不断滚动的流程，每个月都会有变动。

标准委员会最终决定，标准在每年的 6 月份正式发布一次，作为当年的正式版本。接下来的时间，就在这个版本的基础上做改动，直到下一年的 6 月份，草案就自然变成了新一年的版本。这样一来，就不需要以前的版本号了，只要用年份标记就可以了。

ES6 的第一个版本，就这样在 2015 年 6 月发布了，正式名称就是《ECMAScript 2015 标准》（简称 ES2015）。2016 年 6 月，小幅修订的《ECMAScript 2016 标准》（简称 ES2016）如期发布，这个版本可以看作是 ES6.1 版，因为两者的差异非常小（只新增了数组实例的 `includes` 方法和指数运算符），基本上是同一个标准。根据计划，2017 年 6 月发布 ES2017 标准。

因此，ES6 既是一个历史名词，也是一个泛指，含义是 5.1 版以后的 JavaScript 的下一代标准，涵盖了 ES2015、ES2016、ES2017 等等，而 ES2015 则是正式名称，特指该年发布的正式版本的语言标准。本书中提到 ES6 的地方，一般是指 ES2015 标准，但有时也是泛指“下一代 JavaScript 语言”。

3. 语法提案的批准流程

任何人都可以向标准委员会（又称 TC39 委员会）提案，要求修改语言标准。

一种新的语法从提案到变成正式标准，需要经历五个阶段。每个阶段的变动都需要由 TC39 委员会批准。

- Stage 1 - Proposal（征求意见阶段）
- Stage 2 - Draft（草案阶段）
- Stage 3 - Candidate（候选人阶段）
- Stage 4 - Finished（定案阶段）

一个提案只要能进入 **Stage 2**，就差不多肯定会包括在以后的正式标准里面。**ECMAScript** 当前的所有提案，可以在 **TC39** 的官方网站 [Github.com/tc39/ecma262](https://github.com/tc39/ecma262)查看。

本书的写作目标之一，是跟踪 **ECMAScript** 语言的最新进展，介绍 **5.1** 版本以后所有的新语法。对于那些明确或很有希望，将要列入标准的新语法，都将予以介绍。

4. ECMAScript 的历史

ES6 从开始制定到最后发布，整整用了 **15** 年。

前面提到，**ECMAScript 1.0** 是 **1997** 年发布的，接下来的两年，连续发布了 **ECMAScript 2.0**（**1998** 年 **6** 月）和 **ECMAScript 3.0**（**1999** 年 **12** 月）。**3.0** 版是一个巨大的成功，在业界得到广泛支持，成为通行标准，奠定了 **JavaScript** 语言的基本语法，以后的版本完全继承。直到今天，初学者一开始学习 **JavaScript**，其实就是在学 **3.0** 版的语法。

2000 年，**ECMAScript 4.0** 开始酝酿。这个版本最后没有通过，但是它的大部分内容被 **ES6** 继承了。因此，**ES6** 制定的起点其实是 **2000** 年。

为什么 **ES4** 没有通过呢？因为这个版本太激进了，对 **ES3** 做了彻底升级，导致标准委员会的一些成员不愿意接受。**ECMA** 的第 **39** 号技术专家委员会（**Technical Committee 39**，简称 **TC39**）负责制订 **ECMAScript** 标准，成员包括 **Microsoft**、**Mozilla**、**Google** 等大公司。

2007 年 **10** 月，**ECMAScript 4.0** 版草案发布，本来预计次年 **8** 月发布正式版本。但是，各方对于是否通过这个标准，发生了严重分歧。以 **Yahoo**、**Microsoft**、**Google** 为首的大公司，反对 **JavaScript** 的大幅升级，主张小幅改动；以 **JavaScript** 创造者 **Brendan Eich** 为首的 **Mozilla** 公司，则坚持当前的草案。

2008 年 **7** 月，由于对于下一个版本应该包括哪些功能，各方分歧太大，争论过于激烈，**ECMA** 开会决定，中止 **ECMAScript 4.0** 的开发，将其中涉及现有功能改善的一小部分，发布为 **ECMAScript 3.1**，而将其他激进的设想扩大范围，放入以后的版本，由于会议的气氛，该版本的项目代号起名为 **Harmony**（和谐）。会后不久，**ECMAScript 3.1** 就改名为 **ECMAScript 5**。

2009 年 **12** 月，**ECMAScript 5.0** 版正式发布。**Harmony** 项目则一分为二，一些较为可行的设想定名为 **JavaScript.next** 继续开发，后来演变成 **ECMAScript 6**；一些不是很成熟的设想，则被视为 **JavaScript.next.next**，在更远的将来再考虑推出。**TC39** 委员会的总体考虑是，**ES5** 与 **ES3** 基本保持兼容，较大的语法修正和新功能加入，将由 **JavaScript.next** 完成。当时，**JavaScript.next** 指的是 **ES6**，第六版发布以后，就指 **ES7**。**TC39** 的判断是，**ES5** 会在 **2013** 年的年中成为 **JavaScript** 开发的主流标准，并在此后五年中一直保持这个位置。

2011 年 **6** 月，**ECMAScript 5.1** 版发布，并且成为 **ISO** 国际标准（**ISO/IEC 16262:2011**）。

2013 年 **3** 月，**ECMAScript 6** 草案冻结，不再添加新功能。新的功能设想将被放到 **ECMAScript 7**。

2013 年 **12** 月，**ECMAScript 6** 草案发布。然后是 **12** 个月的讨论期，听取各方反馈。

2015 年 **6** 月，**ECMAScript 6** 正式通过，成为国际标准。从 **2000** 年算起，这时已经过去了 **15** 年。

5. 部署进度

各大浏览器的最新版本，对 **ES6** 的支持可以查看kangax.github.io/compat-table/es6/。随着时间的推移，支持度已经越来越高了，超过 **90%** 的 **ES6** 语法特性都实现了。

Node 是 JavaScript 的服务器运行环境（runtime）。它对 ES6 的支持度更高。除了那些默认打开的功能，还有一些语法功能已经实现了，但是默认没有打开。使用下面的命令，可以查看 Node 已经实现的 ES6 特性。

```
// Linux & Mac
$ node --v8-options | grep harmony

// Windows
$ node --v8-options | findstr harmony
```

我写了一个工具 **ES-Checker**，用来检查各种运行环境对 ES6 的支持情况。访问ruanyf.github.io/es-checker，可以看到您的浏览器支持 ES6 的程度。运行下面的命令，可以查看你正在使用的 Node 环境对 ES6 的支持程度。

```
$ npm install -g es-checker
$ es-checker

=====
Passes 24 feature Dectations
Your runtime supports 57% of ECMAScript 6
=====
```

6. Babel 转码器

Babel 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 and 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 默认只转换新的 JavaScript 句法（syntax），而不转换新的 API，比如 `Iterator`、`Generator`、`Set`、`Map`、`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` 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` 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"] } ] ]
  }
}
```


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。

7. Traceur 转码器

Google 公司的Traceur转码器，也可以将 ES6 代码转为 ES5 代码。

直接插入网页

Traceur 允许将 ES6 代码直接插入网页。首先，必须在网页头部加载 Traceur 库文件。

```
<script src="https://google.github.io/traceur-compiler/bin/traceur.js"~>/script>
<script src="https://google.github.io/traceur-com 上一章 10 下一章 js"></script>
```

```
<script src="https://google.github.io/traceur-compiler/src/bootstrap.js"></script>
<script type="module">
  import './Greeter.js';
</script>
```

上面代码中，一共有 4 个 `script` 标签。第一个是加载 `Traceur` 的库文件，第二个和第三个是将这个库文件用于浏览器环境，第四个则是加载用户脚本，这个脚本里面可以使用 `ES6` 代码。

注意，第四个 `script` 标签的 `type` 属性的值是 `module`，而不是 `text/javascript`。这是 `Traceur` 编译器识别 `ES6` 代码的标志，编译器会自动将所有 `type=module` 的代码编译为 `ES5`，然后再交给浏览器执行。

除了引用外部 `ES6` 脚本，也可以直接在网页中放置 `ES6` 代码。

```
<script type="module">
  class Calc {
    constructor() {
      console.log('Calc constructor');
    }
    add(a, b) {
      return a + b;
    }
  }

  var c = new Calc();
  console.log(c.add(4,5));
</script>
```

正常情况下，上面代码会在控制台打印出 9。

如果想对 `Traceur` 的行为有精确控制，可以采用下面参数配置的写法。

```
<script>
  // Create the System object
  window.System = new traceur.runtime.BrowserTraceurLoader();
  // Set some experimental options
  var metadata = {
    traceurOptions: {
      experimental: true,
      properTailCalls: true,
      symbols: true,
      arrayComprehension: true,
      asyncFunctions: true,
      asyncGenerators: exponentiation,
      forOn: true,
      generatorComprehension: true
    }
  };
  // Load your module
  System.import('./myModule.js', {metadata: metadata}).catch(function(ex) {
    console.error('Import failed', ex.stack || ex);
  });
</script>
```

上面代码中，首先生成 `Traceur` 的全局对象 `window.System`，然后 `System.import` 方法可以用来加载 `ES6`。加载的时候，需要传入一个配置对象 `metadata`，该对象的 `traceurOptions` 属性可以配置支持 `ES6` 功能。如果设为 `experimental: true`，就表示除了 `ES6` 以外，还支持一些实验性的新功能。

Traceur 也提供一个在线编译器，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。

上面的例子转为 ES5 代码运行，就是下面这个样子。

```
<script src="https://google.github.io/traceur-compiler/bin/traceur.js"></script>
<script src="https://google.github.io/traceur-compiler/bin/BrowserSystem.js"></script>
<script src="https://google.github.io/traceur-compiler/src/bootstrap.js"></script>
<script>
$traceurRuntime.ModuleStore.getAnonymousModule(function() {
    "use strict";

    var Calc = function Calc() {
        console.log('Calc constructor');
    };

    ($traceurRuntime.createClass)(Calc, {add: function(a, b) {
        return a + b;
    }}, {});

    var c = new Calc();
    console.log(c.add(4, 5));
    return {};
});
</script>
```

命令行转换

作为命令行工具使用时，Traceur 是一个 Node 的模块，首先需要用 npm 安装。

```
$ npm install -g traceur
```

安装成功后，就可以在命令行下使用 Traceur 了。

Traceur 直接运行 ES6 脚本文件，会在标准输出显示运行结果，以前面的 `calc.js` 为例。

```
$ traceur calc.js
Calc constructor
9
```

如果要将 ES6 脚本转为 ES5 保存，要采用下面的写法。

```
$ traceur --script calc.es6.js --out calc.es5.js
```

上面代码的 `--script` 选项表示指定输入文件，`--out` 选项表示指定输出文件。

为了防止有些特性编译不成功，最好加上 `--experimental` 选项。

```
$ traceur --script calc.es6.js --out calc.es5.js --experimental
```

命令行下转换生成的文件，就可以直接放到浏览器中运行。

Traceur 的 Node 用法如下（假定已安装 `traceur` 模块）。

```
var traceur = require('traceur');
var fs = require('fs');

// 将 ES6 脚本转为字符串
var contents = fs.readFileSync('es6-file.js').toString();

var result = traceur.compile(contents, {
  filename: 'es6-file.js',
  sourceMap: true,
  // 其他设置
  modules: 'commonjs'
});

if (result.error)
  throw result.error;

// result 对象的 js 属性就是转换后的 ES5 代码
fs.writeFileSync('out.js', result.js);
// sourceMap 属性对应 map 文件
fs.writeFileSync('out.js.map', result.sourceMap);
```

留言

82 Comments

ECMAScript 6 入门


1 Login

Recommend 26

Tweet

Share

Sort by Best




Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name



Liu Yongjian • 4 years ago

ES7的Async函数，多线程和trait都很期待

45 ^ | v • Reply • Share ›



小胡子哥 • 5 years ago

你写的东西都是一目了然，不管是内容还是布局，真够用心的！


17 ^ | v • Reply • Share ›



6yang ➔ 小胡子哥 • 3 years ago

写的很好，我在window git 下用，每次运行babel xxx.js 要5秒左右才显示结果，有问题？

^ | v • Reply • Share ›



ruanyf Mod ➔ 6yang • 3 years ago

babel确实很慢，只要结果正确就没问题。

^ | v • Reply • Share ›



limingyan ➔ 小胡子哥 • 3 years ago



limingyan • 6 years ago
我看不懂 我装了node后 npm在好几个地方都有 那些命令是怎么回事 直接打开cmd就输入吗 能给我个联系方式吗 我现在乱乱的 对于这个转码器

^ | v • Reply • Share ›



rtyuiuhghjk → limingyan • 2 years ago

因為也曾經看不懂，所以分享一下我的理解 :P。前面大大說的「通过 npm 命令使用」，是否可以說成這個意思：「在 cmd 中用 "npm" 開頭來輸入（npm 的）命令」，像是用 npm 安裝 xxx（假設xxx 是一個套件），就輸入 npm install xxx，至於前面\$符號，應該是代表他處於命令列介面的意思。

如果有概念錯誤也麻煩各位指正啊 ><（現在仍是摸索中）

^ | v • Reply • Share ›



ruanyf Mod → rtyuiuhghjk • 2 years ago

\$ 代表命令行提示符，输入后面的内容即可。

2016-08-12 15:32 GMT+08:00 Disqus <notifications@disqus.net>:

1 ^ | v • Reply • Share ›



Jonas.Gao → limingyan • 3 years ago

刚入门？叫 npm 的文件也许在好几个地方都有。但是单纯使用的话，都是在 cmd 中通过 npm 命令使用（一般就算双击 npm 的批处理你也没啥用）。npm 属于一种包管理器。通过它安装 Babel 的相关组件就可以使用了。npm 会自动处理你安装组件的依赖关系。例如你安装 Babel-cli 的话，那么他会自动安装 cli 所依赖的其他 NodeJS Package。如果你安装的包不是 -g 即全局安装的话，是无法直接在 cmd 中使用的。追加了 -g 参数，把 Babel-cli 安装到全局后，就可以在 cmd 中使用了。具体使用参照上面的文章。暂时这些.....

^ | v • Reply • Share ›



Voyga • 4 years ago

很喜欢老师写的东西，不过作为新手问下，es6的新东西好久才能用啊？

6 ^ | v • Reply • Share ›



jing zhang • 3 years ago

请问 .babelrc 的文件起什么名字，什么名字都可以么

1 ^ | v • Reply • Share ›



ruanyf Mod → jing zhang • 3 years ago

文件名就是 .babelrc 啊

^ | v • Reply • Share ›



jing zhang → ruanyf • 3 years ago

一开始使用mac的sublime2创建并保存，总不成功，提示不可以以 .babelrc 为文件名，后来在 Windows的sublime3上面创建成功，传到mac上就能用了，谢谢大神

^ | v • Reply • Share ›



happy wang • 5 years ago

node -v ==> v0.11.4

在clone nvm到本地之后，cd到nvm的文件夹内 执行了 source nvm.sh 可是之后的use始终是不正常的，不知道是我哪个地方做错了

`npm use 0.11 ==> N/A version is not installed yet`

`npm use 0.11.4 ==> N/A version is not installed yet`

`npm use v0.11.4 ==> v0.11.4 version is not installed yet`

执行 `node --v8-options | grep harmony` 的结果是

`--harmony_typeof` (enable harmony semantics for typeof)

`--harmony_scoping` (enable harmony block scoping)

`--harmony_modules` (enable harmony modules (implies block scoping))

`--harmony_symbols` (enable harmony symbols (a.k.a. private names))

[see more](#)

1 ^ | v • Reply • Share ›



ruanyf Mod ➔ happy wang • 5 years ago

使用 `npm ls-remote` 列出所有可用的版本，然后选择最新版用 `npm use <版本号>` 安装。

1 ^ | v • Reply • Share ›



J.W • 5 years ago

`--harmony` 不是已经包含 `--harmony-generators` 了么？

1 ^ | v • Reply • Share ›



ruanyf Mod ➔ J.W • 5 years ago

谢谢指出，改过来了。

1 ^ | v • Reply • Share ›



see5 • 5 years ago

ES6文章很及时啊，欢迎访问 www.jser.com

1 ^ | v • Reply • Share ›



Justin chan • 9 months ago

很详细，很好

^ | v • Reply • Share ›



友人**C** • a year ago

支持大大！

^ | v • Reply • Share ›



小董君 • a year ago

我的node版本是6.11.1，用es-checker工具检查了一下，已经支持90%的新特性了，😊

^ | v • Reply • Share ›



David Lu • 2 years ago

// 转码前

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

[上一章](#)

[下一章](#)

// 转码后

```
input.map(function (item) {  
  return item + 1;  
});
```

这段最好还是添加个注释，不然很容易让人误会觉得箭头函数就是普通的函数

^ | v • Reply • Share ›



Kevin Liu • 2 years ago

我是来看阮老师的

^ | v • Reply • Share ›



wkliang • 2 years ago

非關技術

「JavaScript 本身也已经被 Netscape 公司注册为商标」

JavaScript 的註冊公司應該是之前的 sun micro，現在的 oracle

^ | v • Reply • Share ›



王译苑 • 2 years ago

阮老师你好！`browserify script.js -o bundle.js`按照教程一步一步做后输出-bash: browserify: command not found，为什么呢？十分感谢！

^ | v • Reply • Share ›



ruanyf Mod ➔ 王译苑 • 2 years ago

Browserify 需要先全局安装。

```
```bash
```

```
$ npm install -g browserify
```

```
```
```

^ | v • Reply • Share ›



幾米 • 2 years ago

<http://eslint.org/docs/rule...>

eslist err...

var & (class this static)

^ | v • Reply • Share ›



onweer • 2 years ago

Nice ~ 想用于我的blog，同时附上原文地址

^ | v • Reply • Share ›



Yefei • 3 years ago

文章提到ES6的支持度，在浏览器、Node.js上多少倍支持都可以都可以理解。但“本机支持ES6的程度”没太理解，执行环境不就是浏览器或Node.js中，本机是个什么概念。

^ | v • Reply • Share ›



ruanyf Mod ➔ Yefei • 2 years ago

[上一章](#)

[下一章](#)

本机就是指你自己正在使用的那台计算机。

^ | v • Reply • Share ›



webkong • 3 years ago

买了一本书，支持一下。大爱阮老师

^ | v • Reply • Share ›



不可能不确定 • 3 years ago

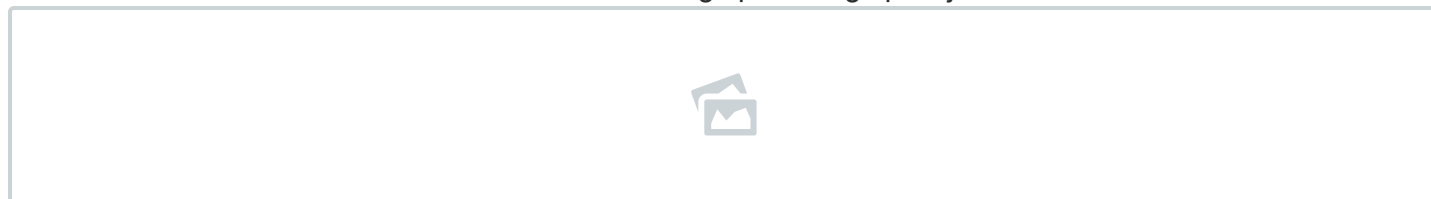
Node 版本管理，感觉 n 比 nvm 好用。

^ | v • Reply • Share ›



jing zhang • 3 years ago

babeljs的官网 <https://babeljs.io/docs/set...> 中点击jade，出现了如图所示的代码，大神能帮忙看下，红框中的代码是要放在哪里么？是要配合构建工具使用，比如gulp，放在gulpfile.js中的么？望指点一二，万分感谢



^ | v • Reply • Share ›



ruanyf Mod → **jing zhang** • 3 years ago

可以参考这篇 <https://www.nathanhleung.co...>

^ | v • Reply • Share ›



Jun Lang • 3 years ago

mark

^ | v • Reply • Share ›



xcatliu • 3 years ago

查了一下才知道 REPL 是什么意思：read-eval-print loop

^ | v • Reply • Share ›



jing2si • 3 years ago

再看一遍，浏览器环境用webpack工具来写es6不错~

^ | v • Reply • Share ›



maochunguang • 3 years ago

很好的文章,辛苦了

^ | v • Reply • Share ›



头头萍 • 3 years ago

```
> curl -o https://raw.githubusercontent.com/number/">/install.sh | bash
```

这个命令似乎执行失败了, curl 7.47.0

看了一下curl手册，-o是指定输出文件：

=====

-o, --output <file>

[上一章](#)

[下一章](#)

Write output to <file> instead of stdout. If you are using {} or [] to fetch multiple documents, you can use '#' followed by a number in the <file> specifier. That variable will be replaced with the current string for the URL being fetched. Like in:

```
curl http://{one,two}.site.com -o "file_#1.txt"
```

or use several variables like:

```
curl http://{site,host}.host[1-5].com -o "#1_#2"
```

You may use this option as many times as the number of URLs you have.

See also the --create-dirs option to create the local directories dynamically. Specifying the output as '-' (a single

dash) will force the output to be done to stdout.

=====

^ | v • Reply • Share ›



ruanyf Mod → 头头萍 • 3 years ago

已经改正。应该是 \$ curl -o- ...

^ | v • Reply • Share ›



火风水 • 3 years ago

traceur.options.experimental = true;会报错，为什么呢？

^ | v • Reply • Share ›



Liyuar Feng • 3 years ago

```
var es6Code = 'let x = n => n + 1';
```

```
var es5Code = require('babel-core')
```

```
.transform(es5Code, {
```

```
presets: ['es2015']
```

```
})
```

```
.code;
```

第三行应该是es6Code

^ | v • Reply • Share ›



ruanyf Mod → Liyuar Feng • 3 years ago

谢谢指出，已经改正。

^ | v • Reply • Share ›




wndfly • 3 years ago

安装node可以用nodenv,蛮好用,和rben 上一章 样 下一章 本共存.不太推荐nvm,这个我之前用ruby时rvm就不太好

就不太好用了。
^ | v • Reply • Share ›

 **xcatliu** → wndfly • 3 years ago
rvm 不好用和 nvm 好不好用有什么关系吗？
^ | v • Reply • Share ›

 星辰梦忆 • 3 years ago
请教：**babel**编译文件时总是出问题，原始文件是什么样，转换后还是什么样，已安装**babel-preset-es2015**，**babel-preset-react**，**babel-preset-stage-2**。
已经改写package.json文件{

"devDependencies": {

"babel-cli": "^6.6.5"

},

"scripts": {

"build": "babel src -d lib"


}


}
已经改写。**babelrc**文件：
{

"presets": ["es2015", "react", "stage-2"],

"plugins": []

}
^ | v • Reply • Share ›

 **ruanyf** Mod → 星辰梦忆 • 3 years ago
Babel默认只转句法，不转新的API。你转一个箭头试试。
^ | v • Reply • Share ›

 星辰梦忆 → ruanyf • 3 years ago
今天早上试了一下**babel**转箭头，还是失败了。
现在使用你在文中提到的另一个工具**traceur**进行转换。
转换前var [a, b, c] = [1, 2, 3];
转换的语句为 traceur --script src/test.js --out lib/test.js --experimental
转换后得到var \$__0 = [1, 2, 3],

a = (\$__1 = \$__0[Symbol.iterator](), (\$__2 = \$__1.next()).done ? void 0 : \$__2.value),

b = (\$__2 = \$__1.next()).done ? void 0 : \$__2.value,

c = (\$__2 = \$__1.next()).done ? void 0 : \$__2.value;
发现其中**Symbol**应该为es6中特有的。不知道是不是选错了**traceur**的版本，该代码在**chrome 48**以及**ff44.0.2**中可以执行，而在**ie11**以下版本中无法执行，是否可以理解为ie中暂时不支持**Symbol**类型。
上一章 下一章

^ | v • Reply • Share ›



ruanyf Mod ↗ 星辰梦忆 • 3 years ago

奇怪，Babel不应该失败，你去网页上转吧 <https://babeljs.io/repl>

^ | v • Reply • Share ›



Frank Lin • 3 years ago

阮大大，ESLint 的配置文件不是 .eslint，应该是 .eslintrc

^ | v • Reply • Share ›



ruanyf Mod ↗ Frank Lin • 3 years ago

谢谢指出，已经改正。

1 ^ | v • Reply • Share ›

Load more comments

ALSO ON ECMASCRIPT 6 入门

编程风格

51 comments • 4 years ago

Tom Bian — mark

修饰器

43 comments • 3 years ago

bpceee — 方法的修饰，target 是 Class 的 prototype，并不是实例。这个时候实例还未创建出来。

SIMD 的用法

3 comments • 3 years ago

dou4cc — Firefox Nightly支持！

异步操作

84 comments • 4 years ago

Qiuleo —

✉ Subscribe Add Disqus to your siteAdd DisqusAdd Disqus' Privacy PolicyPrivacy PolicyPrivacy Policy

