

38 | 编译和打包：通过Webpack、Babel做编译和打包

2022-12-15 石川 来自北京

天下无鱼
<https://shikey.com/>

《JavaScript进阶实战课》

课程介绍 >



讲述：石川

时长 11:57 大小 10.92M



你好，我是石川。

在 JavaScript 从 ES5 升级到 ES6 的时候，在大多浏览器还尚未支持新版的 JavaScript 的时候，很多开发者就想要预先体验相关的功能，但是却愁于没有相关的环境支持。当时，为了解决这个问题，一些面向 JavaScript 开发者的编译器如 Babel.js 就诞生了，它允许开发者按照 ES6 版的 JavaScript 语法来编写程序，然后再将代码编译转化成与 ES5 兼容的版本。

当然后来，随着浏览器对新版 JavaScript 的支持，Babel.js 不单单是解决了 ES6 的使用问题，同时它也加入了很多新的功能，支持更多的编译需求。今天，我们就来看下 JavaScript 开发中用到的 Babel 编译。

JavaScript 的编译

我们知道 JavaScript 是以 ES6 版本作为一个重要里程碑的。它和在此之前的 ES5 版本相隔了 6 年，而在 ES6 问世之后，JavaScript 就保持了每年的更新。所以 ES6 可以被看做是一个“大版本”的更新，里面包含了很多的新功能和语法糖。



面对不同浏览器对 ES6 的支持程度的不一致，有两种处理的办法。一种是**编译**，一种是 **polyfill**。这两者没有明显的界限，但是大致的区别是编译会在运行前先将代码转化成低版本的代码，再运行。而 **polyfill** 则是在运行时判断浏览器是否支持一个功能，只有在不支持的情况下，才使用补丁代码；如果支持，就使用原生的功能。


今天，我们重点说到的就是编译这种方式。**Babel** 在过去很长一段时间，提供了用来帮助开发者提前使用下一代的 JavaScript 版本来编写代码的编译工具，它可以将 **ES6+ 的代码编译成向下兼容的版本**。你可能会问，编译不是浏览器的工作吗？为什么我们说 Babel 是一个编译器呢？

其实 Babel 是一种从源代码到源代码的编译，不是从源代码到机器码的编译，所以为了和浏览器中的 JavaScript 引擎做区分，Babel 也被叫做“**转**”译器（transcompiler 或 transpiler）。Babel 在 2014 年推出后，在很多浏览器尚未提供 ES6 支持的情况下，就让 Web 开发人员能够使用 ES6 和更高版本的新语言功能了。

有些 ES6 的语言特性可以相对容易地转换为 ES5，例如函数表达式。但是有些语言特性，例如 **class** 关键字等则需要更复杂的转换。通常，Babel 输出的代码不一定是开发者可读的，但是可以将生成的代码与源代码的位置进行映射，这在编译后，运行时对问题的排查有很大的帮助。

随着主流浏览器对 ES6+ 的版本支持越来越快，以及 IE 退出了历史舞台，如今编译箭头函数和类声明的需求也大大减少了。但对一些更新的语言特性，Babel 仍然可以提供帮助。与我们前面描述的大多数其它工具一样，**你可以使用 NPM 安装 Babel 并使用 NPX 运行它**。

Babel 可以通过 `npm install --save-dev @babel/core` 来安装，然后直接在代码中导入使用，但是这样的使用会把编译的工作加到终端用户侧，更合理的方式应该是将 Babel 的使用放在开发流程中。

 复制代码

```
1 const babel = require("@babel/core");
2 babel.transformSync("code", optionsObject);
```

在开发的流程中加入 Babel 的方式如下：



复制代码

```
1 npm install --save-dev @babel/core @babel/cli
2 ./node_modules/.bin/babel src --out-dir lib
```

之后，我们可以加入预设（preset）。预设的加入有两种方式，一种是将所用的编译插件都一次性安装；另外一种则是只针对特定的插件，比如箭头函数做安装。

复制代码

```
1 npm install --save-dev @babel/preset-env
2 ./node_modules/.bin/babel src --out-dir lib --presets=@babel/env
3
4 npm install --save-dev @babel/plugin-transform-arrow-functions
5 ./node_modules/.bin/babel src --out-dir lib --plugins=@babel/plugin-transform-a
```

假如我们安装了上述的箭头函数插件，那么在代码中，如果我们使用相关的箭头，Babel 在编译的过程中就会将代码转化成 ES5 版本的样式。在 Babel 内部，通过读取一个 .babelrc 配置文件，来判断如何转换 JavaScript 代码。所以，你可以按需创建想要编译的功能，来进行这些预设，也可以全量使用所有插件，来对所有的功能进行相关的转译。比如在下面这个例子中，就是将一个箭头函数转化为和 ES5 版本兼容的代码。

复制代码

```
1 // Babel输入：ES6箭头函数
2 [1, 2, 3].map(n => n + 1);
3
4 // Babel输出：ES5匿名函数
5 [1, 2, 3].map(function(n) {
6   return n + 1;
7 });
```

尽管现在很多时候，我们已经不太需要转换 JavaScript 的核心语言了，但 Babel 仍然常用于支持 JavaScript 语言的非标准扩展，其中一个就是我们在前面一讲提到的 Flow。Babel 在编译的过程中，可以帮助我们去掉 Flow 的类型注释。除了 Flow 外，Babel 也支持去掉 TypeScript 语言的类型注释。

```
1 npm install --save-dev @babel/preset-flow
2 npm install --save-dev @babel/preset-typescript
```



通过把 Babel 和一个代码打包工具结合起来使用，我们可以在 JavaScript 文件上自动运行 Babel。这样做可以**简化生成可执行代码的过程**。例如，Webpack 支持一个“babel 加载器”模块，你可以安装并配置该模块，以便在打包的每个 JavaScript 模块上运行 babel。那么说到这里，下面我们再来看看 JavaScript 中的打包工具。

JavaScript 的打包

如果你使用 JavaScript 做模块化开发的话，应该对代码打包不会陌生。即使是在 ES6 之前，相关的导入（import）和导出（export）指令还没有被正式引入之前，人们就开始使用这些功能了。

当时，为了能提前使用这些功能，开发者会使用一个代码打包工具以一个主入口为开始，顺藤摸瓜，通过导入指令树查找程序所依赖的所有模块。然后，打包工具会把所有单独模块的文件合并成一个 JavaScript 代码文件，然后重写导入导出指令，让代码可以以转译后的形式运行。打包后的结果，是一个可以被加载到不支持模块化的浏览器的单一文件。

时至今日，ES6 的模块几乎被所有的主流浏览器支持了，但是开发者却仍然使用代码打包工具，至少在生产发布时还是这样的。这么做的原因是为了**将核心功能一次性加载**，这样比起一个个模块单独加载，性能更高，并且可以带来更好的用户体验。

目前市面上有很多不错的 JavaScript 打包工具。其中比较出名的有 Webpack、Rollup 和 Parcel。这些打包工具的基础功能基本上都大同小异，它们的区别主要是在**配置和易用性**上。Webpack 可以算是这几个工具中最元老级的一个了，并且可以支持比较老的非模块化的库。但同时，它也比较难配置。和它正好相反的是 Parcel，一个零配置的替代方案。而 Rollup 相比 Webpack，更加简约，适合小型项目的开发。

除了基础的打包外，打包工具也可以提供一些额外的功能。比如加载的优化，非标模块化插件，更新加载和源代码问题排查等等。下面，让我们一一来看下这些功能。

加载优化

比如很多程序都有多个入口。一个有很多页面的 **Web** 应用，每个页面都有不同的入口。打包工具通常会允许我们基于每个入口或几个入口来创建一个包。



前面我们在讲到前端的设计模式的时候，曾经说过，一个程序除了可以在初始化时静态加载资源外，也可以使用导入来按需动态加载模块，这样做的好处是可以优化应用初始化的时间。通常支持导入的打包工具可以创建多个导出的包：一个在初始时加载的包，和一个或多个动态加载的包。多个包适用于程序中只有几个 **import** 调用，并且它们加载的模块没有什么交集。如果动态加载的模块对依赖高度共享的话，那么计算出要生成多少个包就会很难，并且很可能需要手动配置包来进行排序。

有时，当我们在模块中引入一个模块的时候，我们可能只用其中的几个功能。一个好的打包工具可以通过分析代码，来判断有哪些未使用的代码是可以从打包中被删除的。这样的功能就是我们前面讲过的摇树优化（**tree-shaking**）。

非标模块化插件

打包工具通常有一个支持插件的架构，并且支持导入和打包非 **JavaScript** 代码的模块。假设你的程序包含一个很大的 **JSON** 兼容的数据结构，我们可以用代码打包工具来将它配置成一个单独的 **JSON** 文件，然后通过声明的方式将它导入到程序中。

 复制代码

```
1 import widgets from "./app-widget-list.json"
```

类似的，我们也可以使用打包工具的插件功能，在 **JavaScript** 中，通过 **import** 来导入 **CSS** 文件。不过，这里需要注意的是，导入任何 **JS** 以外的文件所使用的都是非标准的扩展，并且会让我们的代码对打包工具产生一定程度上的依赖。

更新加载

在像 **JavaScript** 这种在执行前不需要预先编译打包的语言里，运行一个打包工具的感觉像是一个预先编译的过程，每次写完代码，都需要打包一次才能在浏览器中执行，这个步骤对于有些开发者而言，可能感觉比较繁琐。

为了解决这个问题，打包工具通常支持文件系统**以观察者的模式**来侦测项目目录中文件的改动，并且基于改动来自动重新生成所需的包。通过这个功能，你通常可以在保存编辑过的文件

后，在不需要手动再次打包的情况下，及时地刷新。有些打包工具还会支持针对开发者的“热更新”选项。每次重新打包的时候，会自动加载到浏览器。



源代码排查

和 Babel 等编译工具类似，打包工具通常也会生成一个源代码和打包后代码的映射文件。这样做的目的，同样是帮助浏览器开发者工具在报错的时候，可以自动找到问题在源文件中的所在位置。

总结

通过今天的学习，我们了解了 JavaScript 中编译和打包工具的前世今生和“成功转型”。

Babel 作为编译器，在很长一段时间能让人们提前使用到 ES6 的功能，在“转型”后，又是在不改变原始的 JavaScript 语言的基础上，让人们可以使用 Flow 和 TypeScript 做类型标注，为其进行编译。

之后，我们学习了 Webpack、Rollup 等代码打包工具，在早期起到了模块化导入和导出的作用，在后期逐渐“转型”为加载优化，提供非标模块化插件的支持，并且提供了代码实时更新打包和热加载的功能。

思考题

今天的思考题，也是作为下面两节课的预习。转型后的 Babel 除了能做到对 Flow 和 TypeScript 进行转译支持，也能支持 JSX 的转译，你知道 JavaScript 中的 JSX 语法扩展的作用是什么吗？

另外一个问题，是我们今天提到除了 Babel 转译外，通过 polyfill 我们也可以解决功能兼容的问题，那么你知道它俩使用场景上的区别吗？

欢迎在留言区分享你的经验、交流学习心得或者提出问题，如果觉得有收获，也欢迎你把今天的内容分享给更多的朋友。我们下节课再见！

分享给需要的人，Ta 购买本课程，你将得 18 元

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 37 | 包管理和分发：通过NPM做包的管理和分发

下一篇 39 | 语法扩展：通过JSX来做语法扩展

更多课程推荐

Vue3 企业级项目实战课

进阶高手的 Vue3+Node.js 全栈开发训练

杨文坚

前阿里前端 leader

前腾讯 IMWeb 团队高级前端工程师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

精选留言

 写留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。