链滴

关于 Babel 与 polyfill 的这些基础知识和区别，你知道吗？

作者：LiuKang

原文链接：https://ld246.com/article/1586408133458

来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20200311.jpg?imageView2/1/w/960/h/540/interlace/1/q/100">
什么是 Babel:smile:
<blockquote>
Babel 官方文档:
</blockquote>
我们知道各个浏览器对 JavaScript 版本的支持各不相同，有很多优秀的新语法都不能直接在浏
器中运行。为了解决这个“沟通不畅”的问题，所以就有了 Babel，Babel 的出现使得我们可以无须
忌的去使用 ES6+ 的语法。
<blockquote>
Babel 是一个 JavaScript 编译器
</blockquote>
这也是为何我们必须使用 ES6+ 语法的前提条件。
<blockquote>
Babel 是一个工具链，主要用于将 ECMAScript 2015+ 版本的代码转换为向后兼容的 JavaScript
语法，
可以使之运行在当前和旧版本的浏览器或其他环境中。
</blockquote>
下面列出的是 Babel 的主要三个功能：
① 语法转换
② 通过 Polyfill 方式在目标环境中添加缺失的特性（通过 @ babel / polyfill 模块）
③ 原始码转换（codemods）
<h2 id="Babel-如何编译">Babel 如何编译</h2>
先看下面这张图：<img src="https://ld246.com/images/img-loading.svg" alt="p1.jpg" dat
-src="https://b3logfile.com/file/2020/04/p1-14e362ee.jpg?imageView2/2/interlace/1/format/
pg">
你会发现 ES6 的语法确实被编译成浏览器可以识别的版本了，你是不是也在问这是怎么做到的
？
<h3 id="babel-编译的阶段">babel 编译的阶段</h3>
<blockquote>
babel 总共分为三个阶段：解析，转换，生成。
</blockquote>
我们需要知道现在 babel 本身是不具备这种转化功能，提供这些转化功能的是一个个 plugin。
以我们没有配置任何 plugin 的时候，经过 Babel 输出的代码是没有改变的。
<img src="https://ld246.com/images/img-loading.svg" alt="640.webp" data-src="https:/
b3logfile.com/file/2020/04/640-9503cb43.webp?imageView2/2/interlace/1/format/jpg"></p

<h3 id="Plugin----transform-的载体">Plugin —— transform 的载体</h3>
<blockquote>
Babel 自 6.0 起，就不再对代码进行转换。现在只负责图中的 parse 和 generate 流程，转换代
的 transform 过程全都交给插件去做。
</blockquote>
例子：
<pre> <code>// 模板字面量</code></
pan> <code>const name </span
= 听风来 ;</code></span
 <code>let hello
span>= `hello ${name}`;</code></
pan> </pre>
上面是一个简单的模板字面量的例子，我们清楚这个是 ES6 的新特性，在不支持 ES6 的运行平
这段代码是会报错的，所以我们需要 Babel 来将其编译成 ES5 的代码。
所以我们需要如下来配置 babel：
<pre> <code>// .babelrc 文件</code>
/span> <code>{ </code>

/span> <code> "plugins
: [</code> <
li> <code>   "transform-es2015-template-lit
rals" // 转译模版字符串的 plugins</code><
span> <code>],</code
 <code> "preset
": ["env",<
span> "stage-2"]</code>
/p> <code>}</code> </l
>
</pre>
preset（即一组预先设定的插件）
<blockquote>
preset: babel 插件集合的预设，包含某些插件 plugin。显然像上面那样一个一个配置插件会非的麻烦，为了方便，babel 为我们提供了一个配置项叫做 persets（预设）。
</blockquote>
当前 babel 推荐使用 babel-preset-env 替代 babel-preset-es201X ,env 的支持范围更广，包含 es201X 的所有语法编译，并且它可以根据项目运行平台的支持情况自行选择编译版本。
<h4 id="plugins-与-presets-同时存在的执行顺序">plugins 与 presets 同时存在的执行顺序</h4>

先执行 plugins 的配置项,再执行 Preset 的配置项；
plugins 配置项，按照声明顺序执行；
Preset 配置项，按照声明逆序执行。

列入以下代码的执行顺序为：

transform-es2015-template-literals
stage-2
env

<pre> <code>// .babelrc 文件</code>
/span> <code>{ </code>
/span> <code>   "plugins
: [</code> <
li> <code>   "transform-es2015-template-l
terals", // 转译模版字符串的 plugins</span
</code> <code>   <spa
>],</code> <code>   </
pan>"presets": [</code></
pan> <code>   [</spa
>"env", {</code>
/span> <code>   // 是否自动
入 polyfill，开启此选项必须保证已经安装了 babel-polyfill</code></
> <code>   // “usage” | “entry
| false, defaults to false.</code> <span
<code>   "useBuiltIns": </spa
>"usage"</code> <cod
>   }], "stage-2"]
/span></code> <code>}</span
</code> </pre>
这里讲一讲 useBuiltIns 配置

无引入 useBuiltIns 配置造成的影响

<blockquote>

我们可能在全局引入 babel-polyfill，这样打包后的整个文件体积必然是会变大的。
</blockquote>

引入 useBuiltIns 配置带来的优势

<blockquote>
通过设置 <code>useBuiltIns</code> 能够把 babel-polyfill 中你需要用到的部分提取出来，需要的去除（按需提取）。<code>useBuiltIns</code> 为 polyfills 应用 @babel/preset-env，选 "usage" | "entry" | false，默认为 false
</blockquote>
useBuiltIns 参数说明：

false: 不对 polyfills 做任何操作
entry: 根据 target 中浏览器版本的支持，将 polyfills 拆分引入，仅引入有浏览器不支持的 polyfil

usage(新)：检测代码中 ES6/7/8 等的使用情况，仅仅加载代码中用到的 polyfills

<h2 id="Babel-相关模块">Babel 相关模块</h2>
以下列举总结了 Babel 的五大核心模块：分别为 babel-core（核心）、babel-cli、babel-nod、babel-polyfill、babel-runtime & babel-plugin-transform-runtime

b3logfile.com/file/2020/04/640-fcb81e40.webp?imageView2/2/interlace/1/format/jpg">
<h3 id="babel-core-核心-">babel-core（核心）</h3>
这个模块是最能顾名思义的了，即 babel 的核心模块。babel 的核心 api 都在这个模块中。也是这个模块会把我们写的 js 代码抽象成 AST 树；然后再将 plugins 转译好的内容解析为 js 代码。</
>
具体怎么工作的这里就不详细说了，因为我也不知道。
<h3 id="babel-cli">babel-cli</h3>
<blockquote>
babel-cli 官方文档：
</blockquote>
babel-cli 是一个通过命令行对 js 文件进行转换的工具。
当然我们一般不会使用到这个模块，因为一般我们都不会手动去做这个工作，这个工作基本都集到模块化管理工具中去了，比如 webpack、Rollup 等。
简单使用（需要先安装 babel-cli）：
<pre> <code>babel test.<spa
>js -o compiled.js
/code> </pre>
<h3 id="babel-node">babel-node</h3>
babel-node 是 babel-cli 的一部分，所以它在安装 babel-cli 的时候也同时安装了。
它使 ES6+ 可以直接运行在 node 环境中。
<h3 id="babel-polyfill-内部集成了-core-js-和-regenerator-">babel-polyfill(内部集成了 core-js regenerator)</h3>
babel 对一些新的 API 是无法转换，比如 Generator、Set、Proxy、Promise 等全局对象，以新增的一些方法：includes、Array.form 等。所以这个时候就需要一些工具来为浏览器做这个兼容。
/p>
<blockquote>
官网的定义：babel-polyfill 是为了模拟一个完整的 ES6+ 环境，旨在用于应用程序而不是库/工。
</blockquote>
babel-polyfill 主要有两个缺点：

使用 babel-polyfill 会导致打出来的包非常大，很多其实没有用到，对资源来说是一种浪费。</li

babel-polyfill 可能会污染全局变量，给很多类的原型链上都作了修改，这就有不可控的因素存。

因为上面两个问题，所以在 Babel7 中增加了 babel-preset-env，我们设置 <code>"useBuiltIns"</code> 这个参数值就可以实现按需加载 babel-polyfill 啦。
<h3 id="babel-runtime---babel-plugin-transform-runtime">babel-runtime & babel-plugn-transform-runtime</h3>
<blockquote>
在使用 Babel6 的时候， .babelrc 文件中会使用 babel-plugin-transform-runtime，而 packag.json 中的 dependencies 同时包含了 babel-runtime，因为在使用 babel-plugin-transform-runtie 的时候必须把 babel-runtime 当做依赖。
</blockquote>
.babelrc 配置：
<pre> <code>{</code> <code> "presets": [</code> <code>   ["env"]</code> <code>],</code> <code> "plugins": [</code> <code>   ["transform-runtime", {</code> <code>   "helpers": false, // defaults to true</code> <code>   "polyfill": false, // defaults to true</code> <code>   "regenerator": true, // defaults to true</code> <code>   "moduleName": "babel-runtime" // defaults to "babel-runtme"</code> <code>   }]</code> <code>]</code> <code>}</code> </pre>
我们在启用插件 babel-plugin-transform-runtime 后，Babel 就会使用 babel-runtime 下的具函数，将一些浏览器不能支持的特性重写，然后在项目中使用。
<blockquote>
babel-runtime 内部也集成了 core-js、 regenerator、helpers 等
</blockquote>
由于采用了沙盒机制，这种做法不会污染全局变量，也不会去修改内建类的原型，所以会有重复用的问题。
现在最好的实践应该是在 babel-preset-env 设置 "useBuiltIns": "usage"，按需引入 polyfill。
<h3 id="三种方案对比">三种方案对比</h3>
<table>
<thead>
<tr>
<th>方案</th>
<th>优点</th>
<th>缺点</th>
</tr>
</thead>

<tbody>
<tr>
<td>@babel/runtime & @babel/plugin-transform-runtime</td>
<td>按需引入, 打包体积小</td>
<td>不能兼容实例方法</td>
</tr>
<tr>
<td>@babel/polyfill</td>
<td>完整模拟 ES2015+ 环境</td>
<td>打包体积过大、污染全局对象和内置的对象原型</td>
</tr>
<tr>
<td>@babel/preset-env</td>
<td>按需引入, 可配置性高</td>
<td>不支持 <code>stage-x</code> 插件，没有利用针对特定浏览器的功能</td>
</tr>
</tbody>
</table>
<h2 id="babel7-的一些变化">babel7 的一些变化</h2>
<h3 id="preset-的变更-">preset 的变更：</h3>
<blockquote>
淘汰 es201x，删除 stage-x，推荐 env
</blockquote>
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。
<h3 id="包名称变化">包名称变化</h3>
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

<h3 id="低版本-node-不再支持">低版本 node 不再支持</h3>
babel 7.0 开始不再支持 nodejs 0.10, 0.12, 4, 5 这四个版本，相当于要求 nodejs >= 6。</
>
还有一些包从其他包独立出来的变化等等
什么是 Polyfill:smile:
<blockquote>
Polyfill 官方文档：
</blockquote>
Polyfill 的准确意思为：用于实现浏览器并不支持的原生 API 的代码、为旧浏览器提供它没有原支持的较新的功能。
比如说 polyfill 可以让 IE7 使用 Silverlight 插件来模拟 HTML Canvas 元素的功能，或模拟 CSS 实现 rem 单位的支持，或 text-shadow，或其他任何你想要的功能。
再例如，querySelectorAll 是很多现代浏览器都支持的原生 Web API，但是有些古老的浏览器不支持，
那么假设有人写了库，只要用了这个库， 你就可以在古老的浏览器里面使用 document.querySelectoAll，
使用方法跟现代浏览器原生 API 无异。那么这个库就可以称为 Polyfill 或者 Polyfiller。
感谢观看，一起遨游知识的海洋~:pray: :pray: