

40 | Polyfill：通过Polyfill让浏览器提供原生支持

2022-12-20 石川 来自北京

天下无鱼
<https://shikey.com/>

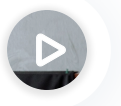
《JavaScript进阶实战课》

[课程介绍 >](#)



讲述：石川

时长 12:28 大小 11.39M



你好，我是石川。

在之前的课程中，我们提到过，用 JavaScript 写的程序不是在统一的环境中运行的。虽然我们
知道现实中存在定义 ECMAScript 规范的组织 TC39，以及编写 HTML5 和 CSS 规范的组织
W3C（万维网联盟），但不同的浏览器厂商对 JavaScript 虚机的实现都会影响我们的程序在
最终执行时的结果。因为虽然标准是存在的，但问题是执行这些标准的公司是独立于标准之外
存在的，这就导致了同样的标准，在不同浏览器厂商和 JS 引擎的理解下，产生了不同的实
现。

今天，我们就看看，如何能在不同的浏览器厂商对同一组较新的 JavaScript 和相关的 Web
API 功能支持程度不一的情况下，通过创建一个 Polyfill，来解决原生支持问题。Polyfill 在英
文中直译过来的意思就是“填缝剂”，我们可以把它理解成一种补丁。

造成原生支持问题的原因

首先，我们来看看浏览器为什么会存在原生支持问题。

原生支持问题其实一直都困扰着开发者，在互联网应用出现的早期，出现最多原生支持问题的便是 IE。这主要是由几方面的原因造成的：

- 第一，IE 早期的版本没有自动更新，而且我们大多数人都没有手动更新软件的习惯，这也就造成了很多的用户在新版本的 IE 提供了更多新功能升级和 bug 修复的情况下，仍然停留在老版本的浏览器上；
- 第二，因为 Windows 的系统早期并不是和电脑捆绑售卖的，也就造成了很多人使用的是破解版的 Windows 系统，为了避免升级时影响盗版的使用，很多人也不会主动更新操作系统，这就进一步造成了人们对系统连带的浏览器版本更新的滞后；
- 第三，企业中使用的 IE 也是浏览器兼容性问题的重灾区。因为越是大的公司，越是“稳定”压倒一切，就越是需要 IT 管理员来集中地更新公司电脑操作系统和软件的版本，而对浏览器这种“病从口入”的连接互联网的窗口软件，更是安全防护的重点，需要集中更新，所以更是大大减缓了浏览器更新的速度。所以早期的网站和 Web 应用开发，主要的问题都集中在 IE6+ 的版本对新版的 JavaScript 原生支持的问题。

如今，随着 IE 退出了历史舞台，向下兼容的问题在逐渐减少，但是问题也不是一下子就完全消失了。因为除了 IE 的问题外，其它的浏览器提供商，如 Chrome、Safari、Mozilla、Opera 等也都是公司。既然是公司，那么它们之间就会存在竞争，从而争取更多的市场份额。

也正因为如此，他们有时会在新定义的 JavaScript 规则还比较模糊的时候加入自己的理解，而不是所有的浏览器都使用同一套新功能发布周期和标准。因为一些公司认为，当 JavaScript 规范中出现新功能时，他们更了解该如何实现，让开发者可以提前体验到新功能，这样也有助于对标准的验证，而另外一些浏览器则并不总是支持一些具有前瞻性的功能。

解决原生支持问题的方法

那么面对不同浏览器对新功能支持的不统一，如果我们想使用较新的 JavaScript 功能，有两个方法：

- 第一，是使用我们在前面提到过的 Babel.js，通过对 JavaScript 的转译，我们可以在提前用到新功能的同时，保证较老版本浏览器的支持；

- 第二个方法，就是使用 Polyfill，Polyfill 作为一个插件，它提供了较新浏览器的功能，但也提供了较旧版本的功能。



前面，我们讲过了代码转译的工具 Babel.js，那我们是不是可以用类似的工具解决所有的原生支持问题呢？

这里，我们需要注意的是，Babel 最主要的能力是让我们提前用上一些新的 JavaScript 语法功能，类似 let、const 这些变量和函数，但是它不能对 JavaScript 中一些新的功能做转译处理。比如转译没法支持数组上的 map 方法函数，也不能对 promise 做转译。这时，我们就需要结合使用到 Polyfill 了。

那我们什么时候该用 Polyfill，什么时候可以用转译器呢？

这里虽然没有明显的界定，但是有个大的原则，你可以参考：**如果你想提前使用一个新的 JavaScript 语法，那么需要用到的是转译器；但如果你想实现一种新的功能或方法，那用到的可能会是 Polyfill。**

另外值得注意的是，虽然 Polyfill 是用 JavaScript 编写的，并且在 JavaScript 引擎中运行，但是它不只适用于 JavaScript 的补丁，HTML 和 CSS 的功能也可以使用 Polyfill 来解决原生支持的问题，甚至我们现在很多手动写的 Polyfill 都是 HTML 和 CSS 相关的补丁。这是因为，Babel 作为 JavaScript 的转译结合 Polyfill 的工具已经解决了大多数语法和功能上的问题。

前面讲到 Babel 的那节课，我们也提到过 ECMAScript 第 6 版是 JavaScript 的主要更新版本，这一版中添加了大量的新功能和语法，所以，如果你想支持 Internet Explorer，或者在其它不支持 ES6 的浏览器中使用最新的 ECMAScript 功能，那么通过使用 Babel，就可以系统性地解决这些问题。但是针对 HTML5 和 CSS3，市面上并没有一个像 Babel 一样的统一工具，因为相关的功能太多，且很多不是都在程序中需要用到的，所以大多是以独立的执行特定任务的补丁的形式出现的。

在标准和浏览器都不断升级的今天，跟踪不断发展的 HTML5 和 CSS 的支持可以说是一项非常艰巨的任务。比如我们想使用 CSS 动画来创造一些页面上的效果，想要知道哪些浏览器支持该功能，对于那些不懂这些代码的浏览器该怎么办，这些问题原本是一个很大的挑战。好在针对这个问题，有一个 html5please.com 的网站，提供了 HTML5 元素和 CSS3 规则的完整列表，并概述了浏览器支持以及列出了每个元素的任何 Polyfill。

Polyfill 的具体实现

但既然我们的重点是讲 JavaScript，我们还是用一个 JS 的例子来看看如何通过 Polyfill 来解决 JavaScript 原生支持问题。Polyfill 的运转机制很简单，当某些功能不被浏览器支持时，它提供了**向下兼容**的方式。

Polyfill 在编写的时候，通常都遵循一个模式，就是我们**先判断脚本想要实现的功能是否已经被当前运行时的浏览器支持了**。如果是，我们就不需要使用 Polyfill 了，如果不支持，那我们就需要 JavaScript 引擎来执行我们定义的补丁。下面，我们可以通过写一个 Polyfill，来更好地了解它的工作机制。

这里我们可以用数组中的 `forEach` 方法来举例。首先，我们来看看 `forEach` 的定义和用途。在定义上，当我们在数组上使用 `forEach` 方法的时候，需要传入一个回调函数。回调函数带有 3 个参数，第一个是必选，代表当前元素的值；第二个是可选，代表当前元素在数组中的索引；第三个也是可选，代表当前元素所属的数组对象。这里面比较常用的是前两个参数，所以也是我们尝试实现的重点。

 复制代码

```
1 array.forEach(function(currentValue, index, arr), thisValue)
```

下面我们再来看看 `forEach` 方法的用途。首先让我们先创建一个数组，我们称之为 `oldArray`。然后里面，我们加入三个地名，纽约、东京、巴黎。之后，让我们再创建一个空数组，`newArray`。我们可以通过遍历第一个数组的方式，将里面的元素推送到第二个数组中。**这就是一个 `forEach` 的简单用例。**

 复制代码

```
1 var oldArray = ["纽约", "东京", "巴黎"];
2 var newArray = [];
3
4 oldArray.forEach(function(item, index) {
5   newArray.push(index + "." + item);
6 }, oldArray);
7
8 newArray; // ['0.纽约', '1.东京', '2.巴黎']
```


通常在写 Polyfill 的时候，我们都会看功能本身是否已经被支持了。为了确定我们当前使用的浏览器是否支持 `forEach` 方法，我们可以使用开发者工具中的控制台，编写一段代码来测试下。



我们可以通过检查 `forEach` 方法是否在数组的 `prototype` 原型上的方式，来看这个功能是否存在。所以，接下来让我们测试下 `forEach` 是否返回 `undefined`，如果返回的结果是 `undefined`，就表示当前浏览器不支持这个方法，也就是说浏览器根本不知道它是什么；如果返回的结果是 `true`，则意味着 `forEach` 不等于 `undefined`，那么 `forEach` 在我当前版本的浏览器上是本地支持的。

复制代码

```
1 Array.prototype.forEach !== undefined;
```

因为 `forEach` 的方法是从 ES6 开始就被支持的了，所以如果你运行上面这一段代码，大概率得到的结果会是 `true`，也就是说你应该可以看到浏览器对 `forEach` 方法的支持。但为了尝试手写 Polyfill 的过程，我们假设当前的浏览器太旧了，它不知道 `forEach` 函数，也未定义 `forEach` 方法，因此它不存在于当前使用的浏览器中。那么让我们手动写个补丁来实现这个功能吧。

下面，我们就可以正式编写 Polyfill 的功能了。首先，我们在建立这个 Polyfill 的时候，第一步要看转参是不是一个函数。既然我们希望这个 Polyfill 能够在所有的数组原型上都能使用，那么我们就需要在 JavaScript 中访问 `Array` 对象，并在其原型上定义一个名为 `forEach` 的新函数方法。

这里有一点需要注意的是，因为我们现在的浏览器是支持 `forEach` 功能的，所以这么做其实就等于覆盖了原生支持的 `forEach` 功能。所以首先，我们需要判断用户传入的是不是一个函数参数。这里，我们可以使用 JavaScript 内置的 `typeof` 方法来实现对参数类型的检查。如果返回的结果不是函数类型的话，程序应该返回一条报错。**这其实就是一种类型检查。**

复制代码

```
1 Array.prototype.forEach = function(callback, thisValue){
2   if (typeof(callback) !== "function") {
3     throw new TypeError(callback + "不是一个函数");
4   }
5   var arrayLength = this.length;
6   for (var i=0; i < arrayLength; i++) {
7     callback.call(thisValue, this[i], i, this);
8   }
}
```

```
9  }
10
11  var oldArray = ["纽约", "东京", "巴黎"];
12  var newArray = [];
13
14  oldArray.forEach( function(item, index) {
15      newArray.push(index + "." + item);
16  }, oldArray);
17
18  newArray; // ['0.纽约', '1.东京', '2.巴黎']
```



如果参数通过了类型检查，那么接下来，我们就要正式写 `forEach` 实现的部分了。

在这个部分中，我们首先需要获取的是**数组的长度**。因为 `this` 代表的是函数在调用时的主体，也就是说我们是通过数组来调用 `forEach` 方法的，所以在这里，我们可以通过 `this.length` 来获得数组的长度。

之后，我们可以通过一个 `for` 循环，来遍历数组中的元素，针对每个元素执行回调函数。在回调函数中，我们可以传入当前元素值、该元素的索引和数组三个参数。这里，为了 `this` 的引用是正确的，我们使用了 `call()` 方法，这样做是为了让 `this` 指向我们传入的正确的第二个参数对象。

同样，我们可以用前面的用例来测试，你会发现我们用来覆盖原始 `forEach` 功能的 `Polyfill`，达到了同样的效果。

总结

通过这一讲的学习，我们知道了如何使用 `Polyfill` 来解决原生支持的问题，也了解了它和转译工具的不同点和互补之处。如果只有使用，没有了解过它们的工作原理，这两个工具是很容易被混淆的。所以我在这里也再次强调下，加深一下理解，转译和 `Polyfill` 的共同点是它们的目的是为了提供原生支持；而它们的区别在于支持的内容和实现的方式。

- 转译解决的主要是**语法的原生支持问题**，它的实现方式是通过将源代码转换成源代码，而不是机器码。
- `Polyfill` 解决的主要是**功能的原生支持问题**，它的实现方式可以是通过在对象原型上增加补丁来支持相关功能。

转译和 **Polyfill** 并不是非此即彼的概念，相反，它们是高度互补的，大多情况下可以结合起来使用。




想必很多同学之前也都使用过 **Polyfill**，希望现在的你不仅会用，还可以掌握设计和编写一个 **Polyfill** 的能力。


思考题

在平时的开发中你有用到 **Polyfill** 的场景吗？你主要使用它来解决哪些问题呢？

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

 生成海报并分享

 赞 0  提建议

© 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 39 | 语法扩展：通过JSX来做语法扩展

下一篇 41 | 微前端：从MVC贫血模式到DDD充血模式

更多课程推荐



天下无鱼

<https://shikey.com/>

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。