ECMAScript——编程语言从构想到标准的历程

Juho Veps¨al¨ainen

阿尔托大学

juho.vepsalainen@aalto.fi

arXiv:2305.01373v1 [cs.PL] 2 May 2023

摘要： *A significant portion of the web is powered by ECMAScript. Asa web technology, it is ubiquitous and available on most platforms natively or through a web browser. ECMAScript is the dominant language of the web, but at the same time, it was not designed as such. The story of ECMAScript is a story of the impact of standardization on the popularity of technology. Simultaneously, the story shows how external pressures can shape a programming language and how politics can mar the evolution of a standard. In this*$_{article,}$ *we will go through the movements that led to the dominant positionof ECMAScript, evaluate the factors leading to it, and consider its evolution using the Futures Triangle framework and the theory of standards wars.*

免责声明

本预印本经 EURAS 2023 许可发布。由于 arXiv 的技术限制，与原文相比，本文做了少量风格调整。

1 引言

根据 [W3Techs, 2023b], ，JavaScript 用于 98.3% 的网站客户端。[W3Techs, 2023a] 估计，其服务器端市场份额约为 2%。尤其是客户端的数据相当可观，体现了 JavaScript 在 Web 开发领域的成功。JavaScript 是标准化及其标准化形式的成功范例；作为一种语言，ECMAScript[1], 是 Web 开发的基石，也是全球最受欢迎、部署基数最大的编程语言[Wirfs-Brock and Eich, 2020]。

正如 Blind 等人[、 2009] 和 [Swann（2010）及 Swann 等人 2010], 所示，标准化的利弊已广为人知。对于 JavaScript 而言，利大于弊，这门编程语言已成为网络经济不可或缺的一部分。可以说，如果没有成功的标准化工作，这一切都不会发生。

[1]命名对商标而言至关重要。因此，标准版本称为 EC- MAScript，而非 JavaScript [Wirfs-Brock and Eich, 2020]。本文中，我们交替使用这两个术语。

Baird 和 Oliver [1977],讨论了 FORTRAN、COBOL 和 PL/1 等早期编程语言的标准化问题；[Hudak 1989] 探讨了函数式语言的相关问题；Stroustrup（2007）和 Stroustrup [讨论了 C 2020]；Urma++ [则从更一般的层面探讨了这一主题。 2017] Wirfs-Brock [从实践角度审视了编程语言标准化问题，并清楚地阐释了标准化机构的运作方式。

[Wirfs-Brock 和 Eich， 2020] 详细回顾了 JavaScript 自 1995年诞生以来的前二十年。这项研究为更深入地探讨标准化如何影响 JavaScript 的发展和普及提供了良好的起点。因此，为了界定研究范围，我们将本文的研究问题正式表述如下：

```
What was the impact of standardization in the evolution of JavaScript?
```

本文部分内容概述自 [Wirfs-Brock 和 Eich 2020], ，我们建议阅读原文，以更深入地了解这一过程，尤其是技术方面的细节。本文的贡献在于运用 [Inayatullah 2008] 的未来三角框架，以及 [Shapiro 等人 1999] 对标准之争的分类，对影响 JavaScript 各个时期的因素进行定性评估。本文为 Web 编程语言领域的研究，尤其是比较研究，开辟了道路。

为了解未来三角和标准之争分类的运作方式，我们先在第 2 节介绍相关理念。随后几节将详细讨论 ECMAScript 及其各个时代，再于第 9 节讨论研究结果，并在第 10 节总结全文。

2 方法论

[Inayatullah, 2008] 的未来三角通过三个方面评估可能的未来：当下的推动力、过去的阻力和未来的吸引力。当下的推动力包括当前趋势、已知政策和近期发展。过去的阻力代表阻碍发展的因素，例如监管或现有投资。未来的吸引力着眼于未来，有助于构想可能的未来。围绕这三个因素进行头脑风暴，可以讨论哪些事情有可能发生、哪些不可能，以及原因 [Fergnani, 2020]。该框架也适用于分析过去，不过即便如此，分析仍带有推测性，并取决于对给定情境的解读。图 1 展示了这三个因素如何组合形成一个三角形。

[Fergnani, 2020] 的未来三角 2.0 通过为每个轴赋予权重，并将其与情景规划相结合，扩展了这一模型。借助扩展后的模型，可以对不同情景进行相对评估。在本例中，这种扩展并没有带来太多价值；但如果你要预测多个合理的未来情景并对其进行详细评估，它就会很有帮助。

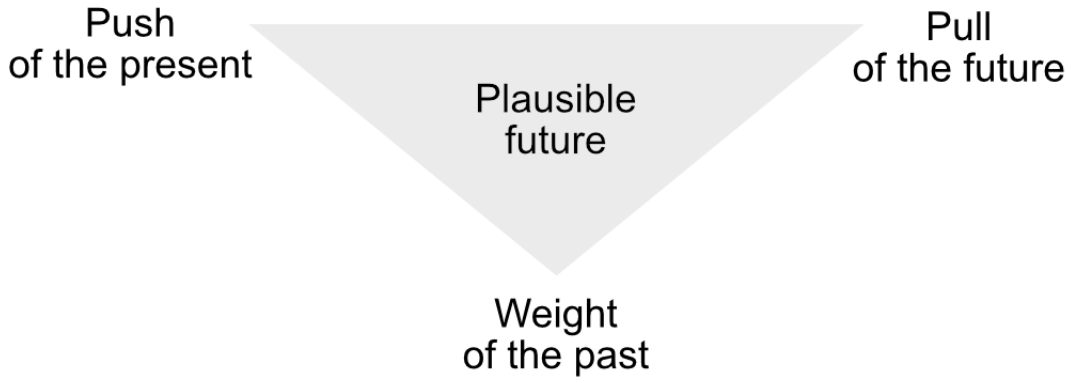

图 1：可能的未来 [Inayatullah， 2008]

[Shapiro 等人, 1999] 对标准战争类型划分的研究提供了另一个评估维度，让我们能够考察 JavaScript 相对于特定时代同期技术的情况。这种分类以与竞争技术的兼容性为切入点，也允许我们考察与相关方法的兼容性。表 1 详细展示了这一模型。

表 1：Shapiro 等人所述的标准之争类型 [Shapiro 等人， 1999]

	竞争技术	
您的技术	兼容	不兼容
兼容	竞争方演进	演进与革命
不兼容	革命与演进	竞争方革命

3 JavaScript 的创建

JavaScript 的故事，是一种语言成为 Web 开发领域主导语言的故事，尽管它最初并非为此而设计。它灵活且足以胜任这一用途，但你也可以说，它至今仍承袭着一些最初的缺陷。随着时间推移，人们对这些缺陷的认识不断加深，这种语言也经过改进，在一定程度上解决了这些问题。周边条件，即对向后兼容性的需求，限制了这些改进；但无论如何，我们今天编写的 JavaScript 与它最初的模样已大不相同。

3.1 背景

自1993年首批 Web 浏览器之一 Mosaic 问世以来，网络已能在不同的桌面环境（Windows、Macintosh、Unix）之间实现可移植性。广泛的支持为跨平台应用程序的开发提供了潜在的平台[Severance， 2012]。正如 [Leprohon， 2017] 所指出的，网页设计师和开发者并没有使用 Java 或遵循面向对象原则的经验，但当时需要一种可在浏览器中运行的小型**脚本语言**。1995年，Netscape Communications 决定为网络引入这种语言。受 Sun Microsystems 及其广受欢迎的 Java 语言影响，人们规定这种语言应当

看起来像 Java，从而排除 BASIC 等某些设计，同时鼓励采用类似 Java 的设计选择 [Wirfs-Brock 和 Eich， 2020]。

创建 JavaScript 首个版本的任务交给了 Brendan Eich。他此前接触过易于使用且具有教学意义的语言，例如 HyperTalk、Logo 和 Self [Severance, 2012]。这些语言的影响体现在 JavaScript 的设计中，这也可能是它包含若干或许不太传统的设计选择的原因。Eich 以十天完成首个语言版本而闻名，时间是 1995年5月[Wirfs-Brock 和 Eich，2020]。

1995年9月，最初版本以 LiveScript 之名随 Netscape Navigator 2.0 的测试版向公众发布。在 3月的正式版中，它更名为 JavaScript 1996 [Wirfs-Brock 和 Eich, 2020]。这一命名选择引发了混淆，因为 JavaScript 与 Java 这种语言并没有太大关系 [Leprohon, 2017]。与完整的语言设计相比，JavaScript 的首个版本并不完善，还包含若干错误；Brendan Eich 在之后的 Netscape Navigator 2.0 多次发布中持续修复这些问题 [Wirfs-Brock 和 Eich，2020]。

3.2 JavaScript 的愿景

最初的愿景是让 JavaScript 与 Java 互为补充，使得用 JavaScript 编写的功能可以与 Java 小程序及其他组件结合使用[Wirfs-Brock 和 Eich， 2020]。颇具讽刺意味的是，从语言设计的角度来看，JavaScript 与 Java 并没有太大关系。不过，JavaScript 在 API 层面继承了 Java 1.0 的若干设计决策，其中一些如今看来值得商榷，例如 Date 模块。JavaScript 1.0 以 C 的语法为蓝本，并从 AWK 中汲取灵感。不过，与 C 不同，JavaScript 语句可以出现在函数体之外，因此更加灵活。此外，两者还有许多其他细微差异，最重要的是，C 使用静态类型，而 JavaScript 使用动态类型 [Wirfs-Brock 和 Eich， 2020]。

我们很容易指出，JavaScript 最初的设计并不完美，这一点连其创始人也承认了 [Eich， 2008]。许多早期问题至今仍困扰着这门语言，因为在编程语言设计中，添加功能很容易，但若要移除它们，却很困难，甚至可能根本做不到，除非进行重大修订。对于一门拥有广泛用户基础的语言来说，发布一个会造成不兼容的版本可能导致意料之外的破坏。本文稍后讨论所谓的 smooshgate 时，会介绍一个具体案例。这个问题也曾出现在从未发布的 ES4 上，我们会在后面的章节中谈及此事。

3.3 JScript——竞争者登场

为了与 Netscape Navigator 保持功能对等，微软为这门语言开发了一款名为 JScript 的解释器 [Wirfs-Brock 和 Eich， 2020]。微软还实现了 Visual Basic Script（VBScript），以此构建自己的技术。为了争取市场份额，不得不支持 JavaScript 虽然令人不悦，却是必要的妥协。

由于当时没有 JavaScript 的官方规范，微软为了保持竞争力，不得不根据 Netscape 的做法对其实现进行逆向工程[Wirfs-Brock 和 Eich， 2020]。在标准问世之前，语言规范体现在 Netscape 对该语言的实现中，而当时这套实现并非开源，也无法供人研究。

3.4 讨论

Netscape 创建 JavaScript 语言，以满足交互式网页的需求，同时为 Sun Microsystems 和 Netscape 带来竞争优势。在 Futures Triangle 中，这体现了**当下的推动力**，其驱动力则是对一种愿景的**未来拉力**：将客户端脚本与服务器端代码（即 Java）结合起来。受各种限制影响，该项目早已带有一些**过去的惯性**，这体现在命名和影响语言设计的高层决策中。

在这三个因素中，眼前的推动力似乎是发展的主要驱动力，因为当时网络仍处于早期阶段，正是影响其发展方向的好时机。与此同时，发展的动力也来自竞争压力，因为 Netscape 并非孤军奋战，正如我们将在下一节中看到的那样。

JavaScript 早期的竞争对手是 JScript。在 [Shapiro 等人提出的 1999] 模型中，这种情况体现了**竞争方演进**： JScript 必须保持与 JavaScript 的兼容性，Internet Explorer 才能争取到市场份额。

JavaScript 的创建催生了一个新市场。如果 Brendan Eich 预先知道后来会发生什么，他很可能会说服 Netscape 在最初的设计上投入更多精力。与此同时，最初的设计已经足够出色，其受欢迎程度超出了当时的预期。JavaScript 的流行度随着网络的流行而提升；随着网络发展，JavaScript 成为开发 Web 应用程序的重要推动力之一。

4 从 JavaScript 到首个 ECMAScript 标准

Netscape 主导 JavaScript 的初期开发后，微软以 JScript 的形式推出了自己的版本，标准化的压力也随之显现。鉴于 Netscape 的领导层对微软的批评颇为敏感，他们希望 Brendan Eich 在 1996 年 着手制定规范，他也确实这样做了：他将最初的 JavaScript 引擎 Mocha 重写为一个名为 SpiderMonkey 的新引擎，并据此提炼出 JavaScript 1.2 [的规范。Wirfs-Brock 和 Eich， 2020]。

自 1995年 JavaScript 项目启动之初，网络就显然需要一个开放脚本标准，微软带来的压力正说明了这一点。Netscape 和 Sun Microsystems 的目标，是找到一个能与微软合作、同时避免微软凭借 VBScript 等技术产品占据主导地位的平台。W3C 和 IETF 曾是早期标准组织的潜在人选，但都不适合 JavaScript。通过私人关系，Netscape 联系了 Ecma，最终找到了合适的组织。 [Wirfs-Brock 和 Eich， 2020]

4.1 TC39首次会议

1996年9月召开的第39技术委员会（TC39）首次会议有30名成员参加。Netscape 的 David Stryker 提议，目标应是制定一份与当时现有实现尽可能少有偏差的规范，并将任何语言扩展留待未来，以反映该语言的当前状态。 [Wirfs-Brock 和 Eich， 2020]

来自微软的 Thomas Reardon 建议委员会避免重复工作，将 HTML 对象模型的标准化交由 W3C 负责，因为 Netscape 和微软的核心功能相似，但 HTML API 各不相同。这些早期

限制促使 TC39 只开发平台无关特性，这至今仍是该组织的核心原则。其目的是制定形式化语言定义，同时为其之外的竞争留出空间。实践证明，这种做法效果良好，竞争也确实出现了，例如在解释器性能方面。[Wirfs-Brock 和 Eich， 2020]

4.2 创建标准的首个草案

[Wirfs-Brock 和 Eich， 2020] 详细描述了制定初始规范时遭遇的早期困难；这是一个充满政治角力与阴谋的故事。本质上，Netscape、Borland 和微软等不同方面各自制定了规范，必须将它们整合为一份共同的规范，作为首个标准。问题在于应以哪份规范为基础进一步完善。由于 Ecma 已习惯使用 Microsoft Word，他们便以微软的初始规范版本为基础开展工作，而没有采用 Netscape 的版本。

规范工作的目标时间定为：1997 年 1 月完成初稿，1997 年 4 月完成终稿。在工作过程中，三份提案中都包含的功能被视为没有争议；其他内容则需要协调。各自独有的功能被移至“提议扩展”，留待日后考虑。另一项重要原则是避免进行会迫使现有应用程序发生变化的修改；这一原则后来成为制定后续标准版本的指导方针。 [Wirfs-Brock 和 Eich， 2020]

4.3 完成初稿

1997年1月10日的 JavaScript 初稿定义了这门语言，其中许多定义至今仍存在于规范中 [Wirfs-Brock 和 Eich， 2020]。这份定义的一项独特之处，是采用伪代码式定义语言来描述语言的特性，从而帮助读者理解语言的工作方式。伪代码介于书面语言与编程之间，是一种绝佳的方式，让人无需学习某种特定语言也能理解内容。

为了赶在4月截止日期前敲定标准，TC39 定期开会。对于任何不明确的边界情况，团队都会编写测试用例，并在当前的运行时中运行，以记录其行为。如果不同实现的行为不一致，就必须达成共识，并明确规定该行为。有些决定至今仍有影响，因此形成了 JavaScript 程序中可见的特定编程惯用法。团队错过了4月的截止日期一个月，于1997年5月2日完成了工作。最终文档成为 ECMA 标准 262，简称 ECMA-262。经过少量编辑后，该文档被提交至 ISO 快速通道流程，标准的第一个版本（ES1）于1997年9月10日发布。 [Wirfs-Brock 和 Eich， 2020]

4.4 讨论

JavaScript 被标准化为 ECMAScript，以应对不断发展的 Web 生态系统所面临的压力。与此同时，标准化也是一种公平竞争的方式，并让开放网络优先发展。另一种选择是由专有平台和技术驱动的网络，而 JavaScript 的标准化并未彻底消除这种可能。对标准的需求是“当下推动力”的鲜明例证，因为它必须反映现有实现的行为，而这些实现代表着“过去的分量

”。标准化工作的成果，则受到早期愿景所带来的“未来牵引力”推动：为网络打造一种可互操作的语言。

在这些因素中，当前的推动力显得尤为重要，因为当时有着明确的标准化需求。未来的牵引力也产生了一定影响，但紧迫的现实状况似乎更具分量。正如我们稍后将看到的，到了 ES4，缺乏明确的愿景成了一个真正的问题。

ECMAScript 标准的创建标志着早期与 JavaScript 相关的标准之争告一段落。它使浏览器厂商能够围绕单一的客户端脚本标准达成一致。

5 从ES1到ES3

ES1 完成后，Netscape 和微软都继续各自开展工作，并为各自的实现添加语言扩展。与此同时，正式的 TC39 会议转变为管理和战略会议，而大部分技术工作则由非正式的技术工作组完成。1997年7月，TC39 就迈向版本2的下一步达成一致，并将其正式完成目标定于1997年12月。1997年9月，会议同意新版本应与早期版本保持向后兼容性。[Wirfs-Brock 和 Eich, 2020]

5.1 ES3——命名和工作方式

鉴于 TC39 和 ISO 的推进速度不同，两个成员构成有所重叠的团队分别为第2版和版本2编写了两份独立的规范草案。为解决这一困境，最终决定 ECMAScript 的下一个版本直接跃升至 3。因此，ES2 从未存在过。 [Wirfs-Brock 和 Eich， 2020]

Netscape 和微软开发的许多语言扩展最终被纳入 ES3 标准，其中一些被删去，另一些则在后续版本中被采用。与 ES1 一样，ES3 的目标是在继续规范已实现功能的同时，增加一些尚不存在于任何地方、且需要开展此前不必要的设计工作的功能。相关工作通过每月举行的面对面会议推进，以解决提案和未决问题，使标准逐步完善。对于国际化（i18n）等更复杂的任务，TC39 组建了专门的工作组。 [Wirfs-Brock 和 Eich， 2020]

与 ES1 相比，这项工作的进展较慢。到1999年春，人们已清楚这项工作无法在6月前完成，但12月前完成仍有可能。3月，大家决定删减或推迟某些功能，以赶上截止期限。初稿于8月完成，在发现并修复了一些小问题后，最终草案于1999年10月提交。经过最后的修改，包括一项重要的向后兼容性修复，ES3 于1999年12月获 ECMA 批准，而下一个版本直到2009年才推出。[Wirfs-Brock 和 Eich， 2020]

5.2 ES3 的采用

正如 [Wirfs-Brock 和 Eich 2020], 所指出的，当时更新 Web 浏览器并不像现在这样普遍，因此直到 2009年 左右，Web 开发者才可以期待 ES3 功能得到广泛支持。后来，工具链和浏览器厂商在很大程度上解决了这一难题。与 1999 或 2009年 相比，如今的发展步伐明显更快，正如我们将在以下

章节中看到的那样，规范如今每年都会更新。在此之前，很长一段时间里情况恰恰相反。

5.3 讨论

从 ES1 到 ES3 的转变，源于**当下的推动力**，需要将浏览器特定语言扩展纳入标准。在标准化过程中，人们也意识到需要规范并添加新功能。同时，还必须**承受过去的重负**，维持向后兼容性。对网页脚本编程的共同愿景，则代表了**未来的牵引力**。

在这些因素中，当下的推动力占据主导地位，因为 Web 平台的演进给标准带来了压力，促使其不断改进、跟上变化，同时提高浏览器之间的兼容性。兼容性对浏览器厂商至关重要，因为它使厂商能够在服务不断增长的网络用户群体时保持竞争力。

6 ES4——那个从未问世的版本

据 [Wirfs-Brock 和 Eich 所述， 2020], 人们对 JavaScript 的看法随着时间而改变：起初，它被视为 Java 的配角，后来却走上了自己的发展道路，Web 开发者也开始意识到 JavaScript 将成为他们唯一需要的语言。随着 JavaScript 的使用日益普及，教育者和布道者开始围绕它出现，并逐渐改变公众对它的看法。除了推广之外，这些人还开发了早期工具链，以支持更大规模场景下的 JavaScript 开发，并创建了数据交换格式 JSON；后者后来成为整个 Web 开发领域的重要组成部分。

随着微软于 1997 年引入 DHTML 和 XMLHTTP，开发交互式动态 Web 应用程序成为可能，无需刷新页面 [Wirfs-Brock 和 Eich, 2020]。尽管此后这一理念只在有限的场景中得到应用，但它一直未为公众所知，直到 2005 年这一技术被命名为 AJAX，并在几年后推动了以 JavaScript 为中心的单页应用程序（SPAs）的兴起 [Severance, 2012, Woychowsky, 2006]。技术发展取代了早期的愿景：当时 Java 小程序与 JavaScript 被结合使用；这也标志着技术格局的一次重大转变。

6.1 ES4——构想 Web 开发的未来

这段背景为理解 ES4 标准化工作为何未能实现提供了良好的起点。到21世纪初，很明显网络是一项具有胜势的技术，而塑造其未来的时机已经到来。与早期的标准化阶段相比，ES4 必须设想未来，因为浏览器厂商最初已无法再提供可供标准化的功能。TC39 内部也存在困惑：究竟是否需要推出一个修正过去错误的破坏性版本，还是应当采取迭代增强的方式。语言的创造者 Brendan Eich 主张推出破坏性版本，而该小组中的一部分人则更倾向于迭代式方法（ES3.1）。为了明确新标准将具有破坏性，它被命名为 ECMAScript 2.0（ES4）。[Wirfs-Brock 和 Eich， 2020]

ES4 的相关工作于1998年初启动，其中充满了从多个方面改进语言的提案 [Wirfs-Brock 和 Eich， 2020]。有趣的是，其中一些功能（例如可选类型注解）至今仍在讨论之中，并且最终可能会进入该语言。

6.2 微软在浏览器大战中的胜利及其对标准化的影响

据 [Wirfs-Brock 和 Eich 所述， 2020], JScript.NET 的发布标志着 ES4 工作走向终结的开端；与此同时，微软赢得了浏览器大战，其 Internet Explorer 浏览器的市场份额超过了 90%，如图 6.2 所示。市场格局的变化意味着，微软可以不再专注于开放网络，转而发展自有技术，以自有专有方案取代现有解决方案。

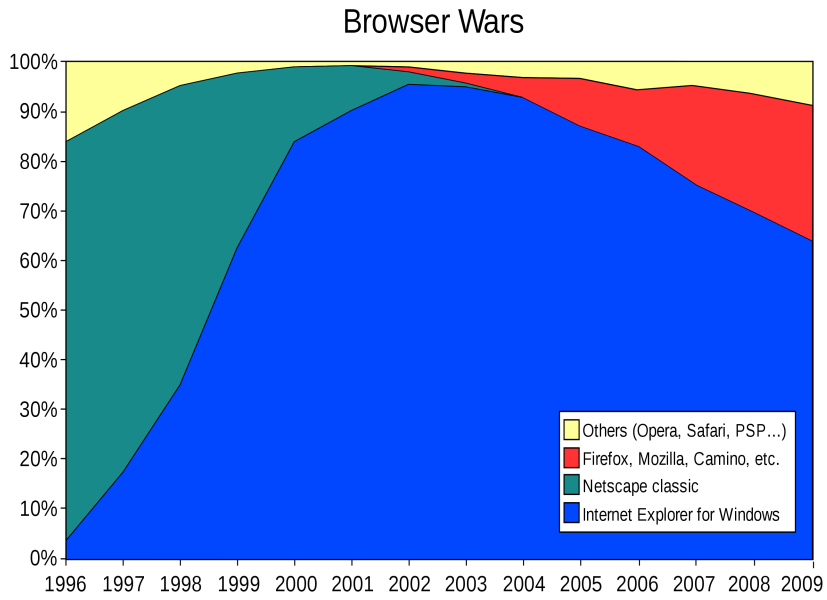

图 2：Wereon 创作，基于 Browser Wars.png 修改，知识共享署名-相同方式共享 3.0

6.3 ActionScript 的影响

同一时期，还有其他几次改进该语言的尝试。其中最值得注意的是 Macromedia 的 Flash 创作工具中的 ActionScript。它是 ES3 的一种方言，并未严格遵循规范。Flash 在 21世纪初尤为流行，被用于实现高度交互式的用户体验。后来，Macromedia 开发了一个虚拟机，并在其上运行拟议中的 ES4 的简化版本，将其作为 ActionScript 3.0。在这项工作完成之前，Macromedia 已被 Adobe 收购，最终成果称为 Adobe Flash。[Wirfs-Brock 和 Eich， 2020]

为推动 ES4 进展，Macromedia（现归 Adobe 所有）于2003年11月加入 Ecma，因为这有助于使 ActionScript 的设计与未来规范保持一致。2004年初，时任 Mozilla 首席技术官的 Brendan Eich 清楚地意识到，开放网络的未来正受到 .NET、Flash 等专有平台的威胁。[Wirfs-Brock 和 Eich， 2020]

6.4 ES3.1 与 ES4

正如 [Crockford, 2012]，所指出的，他加入 TC39 时，委员会中没有使用该语言的成员，因为当时的成员只有浏览器厂商。Crockford 并不相信语言的新版本能够解决任何问题，但他找到了来自微软的盟友 Allen Wirfs-Brock，并说服他相信，采用迭代式的语言设计方法会是更好的选择 [Wirfs-Brock 和 Eich， 2020]。双方的合作促成了另一项提案，即后来所称的 ES3.1；该提案与 ES4 工作并行推进 [Wirfs-Brock 和 Eich， 2020]。

随着 ES4 工作持续推进、时间安排不断调整，作为关键推动者之一的 Adobe 于 2008 年决定退出。Adobe 的退出意味着 ES4 工作告终，也标志着 ECMAScript 协调工作的开始。协调之后，计划是专注于完成 ES3.1，同时合作开展名为 Harmony 的后续项目；该项目不会受 ES4 决策的限制。 [Wirfs-Brock 和 Eich， 2020]

6.5 讨论

ES4 未能成为标准，是标准化可能失败的一个典型例子。在 ES4 期间，TC39 致力于设计一种新语言，以纠正过去的错误，而非像之前那样将现有行为标准化。正如 Crockford 所指出的，ES4 失败的原因在于，标准化机构应当负责规范，而不是设计[Wirfs-Brock 和 Eich， 2020]。设计是适合由社区承担的任务；正如我们稍后将看到的，TC39 已采用一种能够实现这一点的工作方式，使 TC39 可以专注于制定规范，从而避免重蹈 ES4 的覆辙。

与早期那些将现有行为编纂为标准的标准化工作相比，ES4 努力试图重新构想整门语言，并纠正过去的错误（**当下的推动力**）。与此同时，还需要支持过去的功能和现有的浏览器，这构成了**历史的重负**。JavaScript 作为面向网络的编程语言，其整体愿景则代表了**未来的牵引力**。

尽管改进这门语言的需求显而易见，但革命性的做法未能成功；相反，正如接下来的章节所述，标准化工作走上了渐进演进的道路。要记住 JavaScript 所处的背景，尤其是它在网络中的应用，这一点很重要。要为 Web 平台开发应用，就必须考虑用户可用的各种浏览器。要让 JavaScript 在 Web 平台上得到改进，语言的演进就必须考虑这些环境因素。由于网络本身的限制和受众的多样性，保持向后兼容性至关重要。因此，对 JavaScript 来说，渐进演进的做法比彻底改造语言更自然，因为彻底改造可能会危及浏览器支持。从现代视角来看，JavaScript 已成为其他语言的编译目标。稍后讨论 TypeScript 和转译时，我们会详细探讨这一点。

随着 ES4 的出现，标准之争再次拉开序幕，因为网络上开始出现 JavaScript 的替代方案。Adobe 的 Flash 就是一个有力竞争者的典型例子，它代表了一种与 JavaScript 不兼容的竞争技术，因此符合 Shapiro 等人 1999] 模型中 [**竞争方革命**的情形。

Flash 和其他替代方案之所以难以长期与 JavaScript 竞争，原因在于 JavaScript 代码交付给客户端的方式。由于浏览器已经内置了解释器，因此无需额外的运行时环境。而替代方案并非如此，这一额外步骤增加了使用竞争技术的阻力。随着网络技术在开放环境中不断发展，情况

对这些替代方案而言变得更加不利；随着平台逐渐具备相应的功能，Flash 等解决方案也被逐步淘汰。

7 ES5

鉴于 ES4 标准化工作未能成功，ECMAScript 调整了方向，将重点重新放在完成 ES3.1 上，同时规划代号为 Harmony 的后续重大版本。完成后，ES3.1 被称为 ES5；后来，Harmony 计划中的许多特性也逐步纳入 ES6（ES2015）。通过浏览器厂商协作和工作组的努力，ES3.1 于 2009年发布为 ES5，比 ES3 晚了十年；官方 ISO 标准则在完成快速通道流程后于 2011 年发布。 [Wirfs-Brock 和 Eich，2020]

7.1 ES3.1 面临的挑战

2007年，主流浏览器包括 Internet Explorer、Mozilla Firefox、Opera 和 Apple Safari。其中，Internet Explorer 因其 JScript 实现未遵循 ES3 规范而声名不佳，ES3.1 工作组不得不处理这一问题。最终形成的偏差文档有助于厘清哪些问题是 Internet Explorer 独有的，哪些问题普遍存在；这些信息也有助于了解应在哪些方面改进微软的实现，以及在哪些方面需要更清楚地规定标准。 [Wirfs-Brock 和 Eich, 2020]

鉴于一些浏览器已经为 JavaScript 实现了自己的语法扩展，工作组同意采用“4 取 3” 的规则，即将普遍实现的功能或行为纳入标准。大家也明确认识到，标准不应接受可能破坏现有网站的变更。事实证明，“4 取 3”的规则已经足够，因为支持度较低的功能采用率不高，变更这些功能并无大碍。这一观察合情合理：只有一两款主流浏览器采用的功能，不太可能被开发者使用，因为开发者往往倾向于支持多种浏览器，所以这类功能注定比较小众。 [Wirfs-Brock 和 Eich，2020]

7.2 ECMAScript 一致性测试套件

与其他语言相比，ECMAScript 规范必须更加严格，以避免实现特定差异，因此也必须比其他语言定义得更加完善。问题在于，ES5 定稿前后，并没有官方的一致性测试套件，于是微软开发了一个名为 ES5conform 的测试套件。与此同时，Google 发布了自己的 ES3 测试套件 Sputnik，该套件是为其 V8 JavaScript 引擎开发的。2010 年，这两个测试套件合并为 Test262，此后该测试套件成为标准化工作不可或缺的一部分[2]。 [Wirfs-Brock 和 Eich， 2020]

[2]正如 JSCert 所示 [Bodin 等人，2014 年；Gardner 等人， 2015], 通过测试来描述一种语言存在一些问题。JSCert 是一种基于 Coq 的 ECMAScript 形式化规范，在实现该工具的过程中，人们发现 Test262 本身存在若干问题。

7.3 ES5 的主要特性

如前所述，ES5 延续了 ES3.1 的工作，因此这是一次保守的发布。它包含以下主要特性 [Dao， 2020]：严格模式、更多数组和对象方法，以及对 JSON 的原生支持。严格模式旨在让开发者使用语言中更严格的子集，遵循良好实践并避免常见陷阱 [Wirfs-Brock 和 Eich， 2020]。从开发者的角度来看，可以通过使用类似 "**use strict;**" 这样的语句来启用它，该语句会告知解释器，对后续代码块或整个文件采用严格语义 [Wirfs-Brock 和 Eich， 2020]。严格模式是在不破坏 JavaScript 语言的前提下构建其更严格版本的一种方式。新增的数组和对象方法是依据便利性和常见编程模式进行标准化的 [Dao， 2020]。新增的 JSON（JavaScript对象表示法）支持提供了一种数据交换格式，可用于定义，尤其适用于服务器通信 [Dao，2020，Wirfs-Brock 和 Eich， 2020]。这些新特性并未带来翻天覆地的变化，但确实改进了语言。更大的变革则留待下一个版本 ES6。随着 ES5 这一过渡版本的推出，在经历了漫长的版本发布停滞后，标准化工作重新步入正轨。

7.4 讨论

ES5 是一次重要的发布，因为它能够以失败的 ES4 未能做到的方式，应对**过往的重负**。ES4 的失败代表着**当下的推动力**，同时也带来了**未来的牵引力**，因为它塑造了这门语言及其发展的路线图。在 ES5 发布期间，由于引入了一致性测试套件，标准化工作的形态发生了变化。官方测试套件让标准制定者和负责实现标准的人都能更轻松地开展工作。测试套件也成为衡量实现质量及其一致性的有效方式。鉴于 Web 平台的运作方式，这一点尤为重要：它增强了人们对解释器的信心，也帮助开发者在此基础上交付代码。

尽管 ES5 在早期版本的标准发布十年后才得以实现，但这表明 TC39 转而采用迭代式工作方式是多么重要：这种方式优先考虑向后兼容性，而不是重新设计语言并解决其缺陷。ES5 中的许多特性都源于不断发展的 Web 应用程序领域中的实际需求，而事实证明，这也成为 JavaScript 在后续版本中持续演进的方式。

由于对 JavaScript 语法不满，CoffeeScript（2009年）等项目开始涌现 [MacCaw，2012]。这项发展体现了 [Shapiro 等人，1999] 模型中的**革命与演进**，正如我们将在下一节看到的，这种创新推动了标准的发展。

8 ES6 及后续发展

ES4 的失败促成了 ES5 这一小幅增量式发布。更具雄心的功能则留待 ES6（ES2015）实现，由 TC39 [中的 Harmony 项目推动，Wirfs-Brock 和 Eich 负责主导， 2020]。需要注意的是，到 ES6 于 2015 年发布时，网络作为一个行业已经发生了巨大变化；随着社交媒体的兴起，Web 应用程序也开始成为主流。应用程序的复杂性不断增长

每年都在提高，标准也必须应对日益增长的需求，以填补开发者发现的空白。

8.1 ES6 的开发

随着 Harmony 项目在 TC39 中的启动，该组织于 2008 年采用了一种新的工作方式：通过邮件列表或会议收集新想法。对于任何新想法，作者都应将设计写入 TC39 维基，作为一份初步提案，随后向组织成员介绍，再决定放弃或进一步完善。经过这些讨论，TC39 于 2009 年为 ES6 明确了目标，并制定了目标声明，为标准制定提供了框架。该组织采用了负责人模式，由一名成员或一个小组负责某项具体功能，并由其负责不断完善该功能，直至它可以纳入标准，从而避免陷入委员会式设计的困境。[Wirfs-Brock 和 Eich， 2020]

通过初步提案和负责人流程，该组织截至 2011年5月收集了 100 多份初步提案，其中 17 份获得批准。鉴于拟议变更的范围以及对向后兼容性的要求，该组织必须确定如何应对这一问题，最终决定彻底避免破坏性变更。 [Wirfs-Brock 和 Eich， 2020]

直到 2015 年，标准才完成，也正是在这一年，ES6（又称 ES2015）发布了。这是迄今为止对标准最大的一次改动，其中包含类、代理、模块、箭头函数以及许多其他改进[Wirfs-Brock 和 Eich， 2020]。问题在于，在浏览器支持这些特性并且有足够多的用户安装相应浏览器之前，如何使用这些特性。转译器应运而生，解决了这一问题。

8.2 JavaScript 作为编译目标

转译器是一种编译器，它以代码作为输入，并输出代码。换句话说，例如，可以使用转译器将 ES6 代码转换为兼容 ES3 的代码。对于有运行时要求的特性，会提供额外代码使其正常运行。在语法层面，代码会被编译为目标平台所使用的较旧语法。转译器包括 Mozilla 的现已弃用的 Narcissus、Google 的 Traceur（同样已弃用）、由社区维护的流行工具 Babel，以及微软的 TypeScript；后者在 JavaScript 之上提供静态类型检查[Wirfs-Brock 和 Eich， 2020]。

2009年推出的 CoffeeScript 值得特别一提，因为它为 ES6 [中引入的 JavaScript 语法提供了灵感 2012]MacCaw, [。作为一个项目，CoffeeScript 尤其针对 JavaScript 的语法局限，并展示了可能的发展方向。引用 2012], MacCaw 的话， [CoffeeScript 的表达非常简洁，因此与原始的纯 JavaScript 相比，代码量减少了三分之一到一半。随着标准的完善，CoffeeScript 的使用逐渐减少。与此同时，TypeScript 的流行度不断上升，因为它解决了开发者的一项主要顾虑——静态类型，并因此有助于提升软件质量 2014][3]Bierman 等人，。

[3]TypeScript 的流行度和影响力体现在标准化层面，因为有一项与类型注解相关的 ECMAScript 提案。

转译器是社区的重要工具，它们让开发者无需等待即可使用新功能。更重要的是，转译器让开发者能够试验未来可能推出的功能，从而推动语言设计。它们实现了 ES6 的快速部署，并已成为许多开发者所用工具链的一部分。

8.3 SmooshGate——通过提案解决问题的一个例子

正如 2018 年所谓的 SmooshGate 事件所示，当前通过功能提案来解决问题的方式并非没有缺陷，因为如果未经充分考虑就批准功能提案，它们可能会破坏网络 [Bynens，2018]。在 SmooshGate 事件中，Mozilla Firefox 的一个夜间构建版本出现了这种情况：它实现了一项新提出的 **Array.prototype.flatten** 行为，结果至少有一个大型网站在浏览器中无法正常运行。这促使 TC39 在将该功能纳入标准之前更改其命名 [Bynens，2018]。

问题在于，JavaScript 允许为对象添加新方法，而流行的库——尤其是 MooTools——早在 2007年 就采用了这种技术，这给标准化带来了麻烦。MooTools 早已实现了 **Array.prototype.flatten** ，但其实现方式与标准中的不同；更糟的是，这种实现方式破坏了 JavaScript 扩展机制。为了减轻对现有网站的影响，必须为**flatten** 另取一个名称，提案之一是**smoosh**，这便是该事件名称的由来 [Bynens, 2018]。最终选定的名称则是简单的**flat**。

8.4 讨论

尽管 ES6 花了相当长的时间才完成，但从社区的角度来看，它取得了巨大的成功。转译器的出现进一步促成了这一成功，使开发者能够在浏览器支持新功能之前，或用户更新浏览器之前，使用这些功能。作为一次发布，它意义重大，包含了许多标准自诞生之初就一直缺少的功能。许多原本计划纳入 ES4 的功能最终进入了 ES6，此后标准也以每年一次的节奏持续完善。这些年度发布以年份命名（例如 ES2023），让标准得以稳步改进，同时避免了大型发布需要耗费大量时间才能推出所带来的问题。

到了 ES6，TC39 已经找到了良好的运作方式。尽管**历史包袱**意味着不能破坏现有功能，但这一点已广为认可。随着社区不断壮大，越来越多的人希望为语言的发展贡献力量，**当下的推动力**也十分明确，具体表现为大量功能提案。**未来的牵引力**则比以往任何时候都更强：随着 JavaScript 的发展，人们清楚地认识到，它将成为现代网络应用程序的支柱，因此也是现代网络不可或缺的一部分。可以说，事情并非只能以这种方式发展，而且这也与 JavaScript 最初仅仅作为 Java 补充的愿景相去甚远。最终，它发展成了一门完整、独立的编程语言，甚至能够通过 Node.js 或 Deno 等产品取代 Java，应用于后端。

随着 ES6 及后续版本的推出，标准之争的迹象再次显现，TypeScript 等编程语言正在挑战 JavaScript 的霸主地位。在 [Shapiro 等人， 1999]的模型中，这体现了**革命与演进**：新兴语言保留了与 JavaScript 的兼容性

，但 JavaScript 无法与竞争对手实现兼容，尽管标准化工作中的理念可能会反向影响 JavaScript。

9 讨论

作为一种语言，JavaScript 已超越了最初的设计目标。其初版广为人知地是在 90 年代仅用十天创建的，目的是为网站实现客户端交互。JavaScript 的开发为 Netscape Communications 带来了竞争优势，使其得以在浏览器大战初期占据先机。为了迎头赶上，微软被迫在其竞争产品 Internet Explorer 浏览器中实现兼容方案 JScript。逆向工程促使各方要求制定标准，因此成立了技术委员会（TC39），并创建了称为 ECMAScript 的标准化版本。

为回答研究问题“标准化对 JavaScript 的演变有何影响？”，表 2 汇总了主要发展及其背景条件。

标准化对 JavaScript 的影响体现在多个层面。回顾过去，并非所有进展都尽如人意；例如 ES4 曾遭遇挫折，TC39 也花了一段时间才找到有效的工作方式。与此同时，ECMAScript 标准的创建为浏览器厂商提供了良好的协作机制，可以说，若没有这种协作，JavaScript 就不会拥有如今的地位。后来，ES6 降低了更广泛的社区参与这项工作的门槛，并通过每年发布标准快照，使其始终保持实用性。

在标准之争中，JavaScript 走过了一条这样的道路：先是竞争，接着是竞争前的一段平静时期，最后走向革命与演进——挑战者将 JavaScript 视为目标语言，同时在其基础上提供更多功能，以吸引用户采用自己的产品。TypeScript 就是这类语言的一个典型例子：它解决了 JavaScript 的主要痛点之一——类型标注，因此吸引了大量用户。

另一个颇有前景的方向是 W3C 制定的 WebAssembly 标准，它让计算密集型任务无需依赖 JavaScript [Haas 等人， 2017]。WebAssembly 为许多现有编程语言提供了编译目标，因此，随着更多代码能够在其上运行，Web 平台也更具吸引力。

9.1 微软引领的另一种未来

如果微软在 Netscape 之前占领浏览器市场，未来或许会截然不同，因为那样一来，微软就能在 Web 平台上利用 Visual Basic 等现有技术。微软曾尝试沿着这个方向发展，推出 Visual Basic Script 和 ActiveX，但都未能成功。由于 Netscape 抢占了先机，它得以迫使微软跟随自己的步伐。如果双方的角色互换，JavaScript 很可能就不会出现，我们或许会转而使用某种由微软率先开创的技术。

在早期开发阶段，标准化是一项不得不采取的举措，使协作与竞争能够同时进行。可以说，如果微软在网络领域早早取得领先，我们可能会拥有一个基于其技术的标准；或者，也可能走向这样一种局面：专有技术蓬勃发展，而我们所熟知的开放网络将不复存在。

9.2 标准化的成本与收益

尽管达成标准在一定程度上明确了竞争规则，但也为官方规范之外的创新留出了空间，例如扩展功能和性能这类难以量化的特性，正如后文将通过 Google 的 Chrome 浏览器及其 V8 JavaScript 引擎所展示的那样。后来，V8 成为 Node.js 以及其他关键技术的核心，使 JavaScript 得以用于后端开发。如今，许多桌面应用程序也使用 JavaScript 开发，它已成为一种真正无处不在的语言。

如果 Netscape 将 JavaScript 保留为专有技术，并迫使微软不断对其进行逆向工程，那对整个 Web 平台都没有好处，而且很可能会阻碍其发展，因为进入门槛较高，能够进入市场的参与者也会更少。同时，假设微软最终获得了更大的市场份额（事实也确实如此），微软本可以采取相同做法，推动自己的专有解决方案；也可以选择标准化，前提是他们认为标准化的收益大于其弊端。

9.3 标准化使浏览器扩展得以纳入标准

鉴于不同的浏览器厂商都在实现各自的 JavaScript 扩展，早期的标准化工作为将最有用的成果纳入 ECMAScript 标准提供了途径，ES3 的进展便是明证。此外，标准化还为明确此前未定义的行为、统一浏览器行为提供了途径。由此带来的益处是，Web 平台变得更加强大，也更有利于开发者。JavaScript 面临的挑战在于，代码会直接在浏览器中解释执行，而不是采用预先打包的二进制形式；后者本可以缓解这一问题，或者至少将问题转移到别处。

9.4 委员会式设计拖慢了标准化进程

正如 ES4 标准化工作失败这一例子所表明的，推动一门编程语言的全面设计工作可能颇具挑战。尽管修正原始语言设计中所犯错误的目标值得肯定，但在这种情况下，TC39 无力承担如此庞大的工作。TC39 汲取的重要教训是，要保持向后兼容性，避免破坏网络，并更加谨慎地对待所做的更改。重大更改仍有可能实现，但必须更仔细地评估其影响。

9.5 迭代改进让标准化重回正轨

ES5、ES6 及后续版本表明，标准化 ECMAScript 的正确方式是循序渐进。在当前的年度发布模式下，各项功能每年都会按照稳定的周期纳入标准。TC39 采用了行之有效的倡导者模式：这种模式让个人能够专注于特定功能，同时 TC39 则可以从更广泛的角度评估提案，并考虑这些提案之间更大范围的相互影响。

该模型允许在 ECMAScript 之外发生的各种发展进入该语言。CoffeeScript 或 TypeScript 等技术引发的趋势有时会启发相关提案。它们还会通过思想领导力，以更间接的方式影响标准：编程趋势也会影响 TC39 的决策者，并可能使他们更容易接受所提出的变更。这种倾向

开发者转向静态类型和更强类型系统，便是近期的一个典型例子，因为这一转变正给标准施加压力，促使其提高基线。

10 结论

ECMAScript 是编程语言标准化助力其巩固地位的一个很好的例子。可以说，如果没有早期标准化，这门语言或许不会以我们熟悉的形式存在，我们在 Web 平台上也可能会使用其他编程语言。尽管标准化过程中曾出现失误和困难，JavaScript 已证明足以胜任 Web 应用程序开发，而且早已超越最初的设计范围；最初，它被设想为 Java 这门“真正的”编程语言的补充。

微软推出的 JavaScript 方言 TypeScript，则从另一个角度说明了标准化问题。尽管 TypeScript 因为能为用户带来显著益处而日益流行，但它并没有语言标准。缺乏标准化让这门语言能够快速发展；例如，为了改进语法或加强类型检查，新版本发布时打破现有行为的情况并不少见。至少就目前而言，可以说 TypeScript 没有标准是件好事，因为标准化会影响它作为一项技术的发展速度。同时，缺乏标准也使第三方围绕 TypeScript 创建工具变得更加复杂，不过迄今这还不是一个主要问题。

已有早期迹象表明，随着与类型标注相关的提案得到审议，TypeScript 可能会对 ECMAScript 产生影响。尽管 ECMAScript 可能会变得更加严格地采用类型，但 TypeScript 似乎不太可能彻底消失；与此同时，ECMAScript 层面的改动也能为其提供更好的编译目标，因为这样在转译过程中便能保留更多类型信息。这反过来可能带来其他好处，例如在执行期间提供更强的类型保证，并且由于所需的类型推断更少，可能获得更好的性能。

TypeScript 缺乏标准化这一点本身就是一个很好的研究课题。更广泛的问题涉及一般的编程语言，因为并非所有流行语言（例如 Perl 或 PHP[4], ）都有官方标准。问题在于，与 ECMAScript 及其流行度所体现的标准驱动模式相比，非标准化模式有哪些优势和缺点?

从更宏观的层面来看，ECMAScript 的案例展现了标准化的力量：它促使竞争者展开合作，并共同开拓了一个庞大的市场。ECMAScript 已成为现代网络不可或缺的一部分，也是推动 Web 应用程序发展的关键技术之一。数据、人工智能或机器学习等相关领域的新兴主题，或许也能带来类似的益处。随着标准不断制定，新的、有时出乎意料的合作方式也随之出现，同时技术的基准水平不断提高，使竞争与创新得以在标准化技术的范围之外展开；而这些成果又会随着时间推移反哺标准。

[4]PHP 没有官方标准，而是采用事实标准，背后也没有标准化组织。

参考资料

[Baird 和 Oliver， 1977] Baird, G. N. 和 Oliver, P.（1977）。编程语言标准——谁需要它们。技术报告，自动数据处理设备选型办公室（海军），华盛顿特区。[Bierman 等， 2014] Bierman, G.、Abadi, M. 和 Torgersen, M.（2014）。理解 TypeScript。载于 ECOOP 2014——面向对象编程：第 28 届欧洲会议，瑞典乌普萨拉，2014年7月28日至8月1日。论文集 28，第 257–281 页。Springer。[Blind 等， 2009] Blind, K. 等（2009年）。标准化作为创新的催化剂。技术报告，伊拉斯谟管理研究院（ERIM）。[Bodin 等， 2014] Bodin, M., Chargu´eraud, A., Filaretti, D., Gardner, P., Maffeis, S., Naudziuniene, D., Schmitt, A. 和 Smith, G.（2014）。可信的机械化 JavaScript 规范。《ACM SIGPLAN 通讯》，49(1)：87–100。[Bynens， 2018] Bynens, M.（2018）。SmooshGate 常见问题 - Chrome 开发者 — developer.chrome.com。https://developer.chrome.com/blog/smooshgate/。[访问日期：07-Mar-2023]。[Crockford， 2012] Crockford, D.（2012）。QCon 旧金山 2012 大会上 JavaScript 的现状与未来。https://qconsf.com/sf2010/dl/qcon-sanfran-2009/slides/DouglasCrockford_TheStateAndFutureOfJavaScript.pdf。[访问日期：07-Mar-2023]。[Dao， 2020] Dao, C.（2020）。JavaScript 的本质与演变。学士学位论文，奥卢应用科学大学。[Eich， 2008] Eich, B.（2008）。流行度。https://brendaneich.com/2008/04/popularity/。 [访问日期：07-Mar-2023]。[Fergnani， 2020] Fergnani, A.（2020）。未来三角 2.0：将未来三角与情景规划相结合。foresight。[Gardner 等， 2015] Gardner, P., Smith, G., Watt, C. 和 Wood, T.（2015）。可信的 JavaScript 机械化规范：一年之后。载于《计算机辅助验证：第 27 届国际会议》，CAV 2015，美国加利福尼亚州旧金山，2015年7月18日至24日，论文集，第一部分 27，第 3–10 页。Springer。[Haas 等， 2017] Haas, A., Rossberg, A., Schuff, D. L., Titzer, B. L., Holman, M., Gohman, D., Wagner, L., Zakai, A. 和 Bastien, J.（2017）。借助 WebAssembly 提升网络运行速度。载于《第 38 届 ACM SIGPLAN 编程语言设计与实现会议论文集》，第 185–200 页。[Hudak， 1989] Hudak, P.（1989）。函数式编程语言的构思、演进与应用。《ACM 计算调查（CSUR）》，21(3)：359–411。[Inayatullah， 2008] Inayatullah, S.（2008）。六大支柱：转型的未来思维。《foresight》，10(1)：4–21。

[Leprohon, 2017] Leprohon，M.-A.（2017）。ECMAScript 6 与 JavaScript 的演进：深入了解这门语言的新特性。学士学位论文，Metropolia Ammattikorkeakoulu。[MacCaw, 2012] MacCaw，A.（2012）。《CoffeeScript 小书：JavaScript 开发者构建更出色 Web 应用的指南》。“O’Reilly Media, Inc.”。[Severance, 2012] Severance，C.（2012）。JavaScript：用 10 天设计一门语言。《Computer》，45（2）：7–8。[Shapiro 等人, 1999] Shapiro, C.、Varian，H. R.、Carl，S. 等人（1999）。《信息规则：网络经济的战略指南》。哈佛商业出版社。[Stroustrup, 2007] Stroustrup，B.（2007）。在现实世界中为现实世界演进语言：C++ 1991-2006。载于《第三届 ACM SIGPLAN 编程语言史会议论文集》，第 4–1 页。[Stroustrup, 2020] Stroustrup，B.（2020）。在拥挤且不断变化的世界中蓬勃发展：C++ 2006–2020。《ACM 编程语言论文集》，4（HOPL）：1–168。[Swann, 2010] Swann，G. P.（2010）。标准化经济学：最新进展。英国商业、创新与技能部（BIS）报告。[Swann 等人, 2010] Swann，G. P. 等人（2010）。国际标准与贸易：实证文献综述。《经合组织贸易政策论文》，（97）。[Urma, 2017] Urma，R.-G.（2017）。编程语言的演进。技术报告，剑桥大学计算机实验室。[W3Techs，2023a] W3Techs（2023a）。网站 JavaScript 使用统计与市场份额，2023年2月 — w3techs.com。 https://w3techs.com/technologies/details/pl-js。[访问日期：27-Feb-2023]。[W3Techs，2023b] W3Techs（2023b）。JavaScript 作为网站客户端编程语言的使用统计，2023年2月 — w3techs.com。 https://w3techs.com/technologies/details/cp-javascript。[访问日期：27-Feb-2023]。[Wirfs-Brock, 2016] Wirfs-Brock，A.（2016）。编程语言标准化：参与模式。载于第五届亚洲程序模式语言会议。[Wirfs-Brock 和 Eich, 2020] Wirfs-Brock，A. 和 Eich，B.（2020）。JavaScript：前 20 年。《ACM 编程语言论文集》，4（HOPL）：1–189。[Woychowsky, 2006] Woychowsky，E.（2006）。AJAX：使用异步 JavaScript 和 XML 创建网页，第 8 卷。普伦蒂斯·霍尔，上萨德尔河，新泽西州，美国。

表 2：JavaScript 的发展阶段

Era	利益相关方	推动 当前	拉动 未来	权重 of 过去	标准 war 竞争对手的演进	结果
创建 （1995—1996 年）	Netscape 和 微软	需求 of the 交互式 web	愿景 of 客户端 com- 结合 服务器端 code	高级 兼容 性 with Java	方案	首次实- 现- 施， 首个竞- 争者 旨在 for com- 兼容性
标准化 化 （1996—1997）	初始 TC39 成员	The need for a 标准	愿景 for an 互操作 性 语言 用于网络	行为 of the 现有的 实现- 方式	已结束 due to 对齐	创建 of EC- MAScript 标准 和 TC39
源自 ES1 to ES3 (1997-1999)	TC39 mem- bers	集成 浏览器特定的 特定 扩展	共享 愿景 of 脚本编写 在网络上	向后 兼容 性	None	改进后的 标准
ES4 (1999-2008)	TC39 mem- bers	修正 语言	愿景 脚本编写 在网络上	向后 兼容性- 兼容性	竞争对手的演进- 解决方案	失败， 改革— tion of TC39, Harmony
ES5 （2008-2015）	TC39 mem- bers	失败 ES4	路线图 受.....影响 失败 ES4	后退 兼容性 能力	革命 相对于 演进	成功 发布 of ES5, 改进后的 way of 工作 为 TC39
ES6 and 超越 （2015 年至今）	TC39 mem- bers 和网络 社区	特性 提案 from the commu- nity	清除 vi- sion of JavaScript 作为一部分 of the 现代 web	向后 兼容- 性	革命 对比 演进	冠军 模式， 年度 快照