

# 50万行代码不敢交给AI? TypeScript之父直言：它就像是个“高级复读机”

CSDN 2026年1月30日 18:54 江苏



编译 | 郑丽媛

出品 | CSDN (ID: CSDNnews)

近日，TypeScript 与 C# 的发明者、微软技术研究员 Anders Hejlsberg 表示，现有编程语言之所以更适合 AI 编程，并不是因为它们更“先进”，而是因为它们拥有最大的训练数据集。

而对于 AI，他给出了一个直接的评价：

当前的大模型，本质上更像是一个“把别人做过的事情重新吐出来，在此基础上做一些简单推演”的高级复读机。

这番表态来自他与 GitHub 研究顾问 Eirini Kalliamvakou 的一次对谈。此次访谈中，Hejlsberg 不仅分享了 TypeScript 团队内部使用 AI 的实际经验，也回顾了 TypeScript 的发展历程，并展望了它未来可能的演进方向。



## // 01

TypeScript 7.0 核心升级：原生编译器登场，性能提升 10 倍

即将到来的 TypeScript 7.0，最重要的变化之一，是引入原生编译器。目前该版本已经进入预览阶段。

Hejlsberg 表示，原先的 TypeScript 编译器是用 TypeScript 自己写的，并运行在 V8 JavaScript 引擎之上。但随着项目规模和使用场景不断扩大，这种实现方式在性能上已经明显吃紧：“我们很快就发现，原生编译器能让性能提升 10 倍——其中一半收益来自原生代码本身，另一半则源于对共享内存并发特性的充分利用。”

看似“从头重写”更干净，但在 TypeScript 的场景下，这几乎是不可行的。

Hejlsberg 解释说，TypeScript 的类型检查器规模巨大、逻辑极其复杂，而且存在大量只体现在代码语义行为中、并没有文档化的隐式规则：

“TypeScript 拥有一个体量庞大、逻辑复杂的类型检查器，其诸多行为逻辑仅体现在现有代码的精准语义中，并无其他文档或形式可完整复刻。”

如果不是逐函数、逐逻辑地移植，而是采用“等价实现”的方式，那么哪怕结果只出现极小差异，都会为用户迁移时引发大量长尾问题。所以，这款全新原生编译器的设计目标非常明确：输出必须和旧编译器完全一致，甚至要包括那些历史遗留的“怪癖”。

## // 02

### 语言选型引争议：弃 Rust、舍 C#，最终选 Go

在为原生编译器选择实现语言时，TypeScript 团队的决定一度引发了不小的争议。

Hejlsberg 透露，迁移的技术需求直接就排除了 Rust：Rust 不支持团队移植所需要的循环数据结构，同时也没有自动垃圾回收（GC），而这些在移植过程中是刚需。

团队最初也对 C# 做了相关试验，但最终还是选择了 Go 语言，因为“Go 与 JavaScript 的语法和设计思路高度相似”。

这一决定让 C# 社区倍感不解，不少开发者质疑：

“作为 C# 的发明者，Hejlsberg 为何不选用你自己设计的语言，顺便提升一下 C# 的影响力？”

Hejlsberg 并没有正面回应这个问题，只是表示：“确实有很多人觉得，我们应该选另一种语言。但我坚信我们选对了工具，过去一年的实践已经证明了这一点。”

## // 03

### 用 AI 做编译器移植？现实并不理想

在 AI 话题上，Hejlsberg 的态度同样相当冷静。他透露，最初团队也曾试图用 AI 完成从 TypeScript 到 Go 的代码迁移工作，但“效果很不理想”。

他直言，代码迁移需要绝对确定的输出结果——团队要迁移 50 万行代码，且要求新代码与原代码的执行逻辑完全一致。但如果让 AI 直接翻译代码，输出内容很容易出现“幻觉”，哪怕只是一点点，也意味着你必须逐行人工检查，反而得不偿失。

这个观点，也与众多开发者对微软 Visual Studio 的吐槽形成了呼应：微软此前弃用了确定性强的 .NET 升级助手，转而推出基于 Copilot 的升级工具，而后者的非确定性输出也让开发者叫苦不迭。

在 Hejlsberg 看来，AI 在代码迁移中的正确打开方式，并非直接翻译代码，而是“让 AI 生成辅助迁移的工具程序”：通过 AI 开发的工具程序，执行后能输出确定的结果，真正为开发提效。此外，他也认可 AI 的技术价值，并表示 TypeScript 的语言服务（负责代码语法检查、修复建议的核心功能）正在大幅适配 AI 技术，“在这个场景下，AI 能发挥出远超人工的效果”。

与此同时，团队也找到了 AI 的合适应用场景：在完成原生编译器的初始迁移后，需要将旧代码库中新增的 PR 迁移至 Go 代码库——而在这个工作中，“AI 的使用效果相当不错”。

## // 04

### TypeScript 的未来：语言不会激进更新，但工具链会彻底改变

谈到 TypeScript 的未来，Hejlsberg 表示，它仍将遵循一贯路线：先跟随 JavaScript 的标准化进程，再在其之上补充必要的类型系统特性。

作为 JavaScript 的超集，TypeScript 的发展本身也在反向影响 JavaScript 的标准化进程，因此，不要期待 TypeScript 语言本身出现激进变化——在 Hejlsberg 看来，TypeScript 未来最大的变革，将发生在工具链层面。

“我曾经觉得 IDE 已经是终极形态了，但 AI 的出现，彻底改变了这一切。”在他看来，如今 AI 不再只是 IDE 中的一个辅助插件，反而变成了开发者需要监督的核心工具，甚至“不再需要传统意义上的 IDE 作为载体”。

但 Hejlsberg 也补充道，AI 工具仍需要语言服务的底层支持，这也是 MCP 等机制愈发重要的原因——将语言服务与 MCP 打通，让 AI 能够直接提出语义级、结构级的问题和修改建

议。

“AI 需要具备等同于 IDE 能做的那些能力，但要以 LLM 或 Agent 的方式来完成。这将会彻底改变开发工具的形态。”

## // 05

### TypeScript 的诞生：并不是想“发明一门新语言”，为修复 JavaScript 的痛点而生

最后，在此次完整访谈中，Hejlsberg 还讲到了 TypeScript 的起源故事。

这门语言的最初构想，来自微软的 Outlook Web 团队。彼时该团队正使用一款名为 Script# 的工具，通过 C# 编写代码并编译为 JavaScript，以实现在浏览器中的运行。

Hejlsberg 看到了这一思路的价值，但并未选择基于 C# 开发，而是决定打造一款 JavaScript 的类型化超集——他的核心初衷是：“通过扩展 JavaScript 的能力，我们并非要创造一门全新的语言，而只是想修复它本身存在的问题。”

如今，TypeScript 已跻身主流编程语言之列，但另一方面，微软将其编译器从自身语言迁移至 Go 的举动，也在无形中承认了它在性能层面的物理上限。正如 RedMonk 分析师 Stephen O’Grady 去年的疑问：

“如果一门语言连自己的编译器都跑不动了，这会不会反过来影响人们对它的看法？”

而这个问题，或许还需要时间来回答。

原文链接：<https://devclass.com/2026/01/28/typescript-inventor-anders-hejlsberg-ai-is-a-big-regurgitator-of-stuff-someone-has-done/>

#### 推荐阅读：

[2026 奇点智能技术大会上海站官宣！解码 AI Agent、世界模型与氛围编程等新范式](#)

[为没有 Linux 的一天做准备！Linux 社区敲定接班预案：若维护者不愿干就立刻定替代人选，这事绝不能拖](#)

[多模态和编程能力可以兼得吗？Kimi 新模型 K2.5 实测](#)

未来没有前后端，只有 AI Agent 工程师。

这场十倍速的变革已至，你的下一步在哪？

4月17-18日，由CSDN与奇点智能研究院联合主办「2026奇点智能技术大会」将在上海隆重召开，大会聚焦Agent系统、世界模型、AI原生研发等12大前沿专题，为你绘制通往未来的认知地图。

成为时代的见证者，更要成为时代的先行者。

奇点智能技术大会上海站，我们不见不散！

SITS 2026

# 奇点智能技术大会

Singularity Intelligence Technology Summit

4月17-18日·上海

 <b>王炳宁</b> 腾讯微信搜索 AI 算法 研究方向负责人， 专家研究员	 <b>张俊林</b> 新浪微博首席科学家 及 AI 研发部负责人	 <b>邓金秋</b> 京东定价算法负责人	 <b>陆承掇</b> 小红书 AI 搜索生成 算法负责人	 <b>许辰人</b> 北京大学博雅 长聘副教授	 <b>宫叶云</b> 微软亚洲研究院人工 智能推理组负责人
--	---	--	---	---	--

扫码领取大会资料

大会合作咨询