Rust & 前端工具链

结合

🐢 JS 编译工具慢是原罪?

单线程语言 | 抽象 AST | 众多插件 | 源代码映射 | 兼容性转换 | I / O 操作

🐆 Turbopack、SWC 为什么这么快?

1. Rust 语言本身：设计用于处理大量的并发和高性能的工作负载，编译器优化后性能接近 C 的水平
2. 并行编译提速：可以并行处理任务，充分利用多核心处理器的性能，尤其在处理大型项目时
3. 优化的算法和数据结构：高度优化的机器代码和低层级增量计算引擎，可以缓存到单个函数的级别
4. 少量的 I/O 操作：尽量减少了 I/O 操作，因为 I/O 操作通常是造成编译器慢的一个重要原因

Rust & Node 模块

结合

高级别的抽象、低级别的操作、极致压缩硬件的性能

高性能计算	图像处理、机器学习、复杂的数学计算、游戏开发 在需要大量计算或处理大量数据的场景中很有优势
系统级操作	访问文件系统、网络、硬件设备、管理内存、浏览器引擎 Rust 的系统编程特性使其更安全、高效的访问操作系统资源
并发和多线程	CPU 密集型任务、数据库高并发读写 很好的支持并发和多线程，让 Node 可获得真正的并行计算能力
增强应用的安全性	加密和区块链技术、物联网设备、网络编程 由于内存安全特性，使用 Rust 写的模块可以极大地减少内存错误和数据竞争等问题