2025-05-30

Tinymist 2024

Review of the Tinymist in 2024.

Contents

2025-05-23

Tinymist 2024 - 语言服务器部分

关于 tinymist，一个 typst 的语言服务器的开发思考。

1.1 LSP 引擎库

在 lsp 引擎上，nvarner 的选择是 tower-lsp，但是这个库事实上并没有尊重 lsp 协议（2024 年的时候，tower-lsp 的情况如此）。lsp 在时序上希望你能保证按顺序处理请求，而 tower-lsp 收到请求上会乱序触发上层 service 的函数。这会导致 language server 状态在启动后一段时间与编辑器状态 desync。tower-lsp 的这一做法也使得允许上层 service 有一些“fancy”的写法，直接导致 typst-lsp 需要完全重写。

rust-analyzer 是怎么做的呢。rust-analyzer 使用了 lsp-server，这是一个底层完全同步的 lsp 引擎。每当有一个请求到来，都会触发一个获得 `state: &mut State` 的 handler。

一位群友为 nix 写的 nil 用了这位群友自研的 async-lsp。其接口要比 lsp-server 整洁和 neat 的多。每当有一个请求到来，async-lsp 也会触发获得全局可变状态的 handler，区别是这个 handler 是 async 的。

我是希望使用 async-lsp 的。在接入的过程中，再一次挖掘到了 rust 的混沌之处。我们做一个表格，lsp-server，async-lsp，tower-lsp 的区别如下：

Name	Order to Accept Requests	Type of Handler
tower-lsp	Out of order	`Fn() -> Fut<Req>`
lsp-server	Sequential	`FnMut() -> Req`
async-lsp	Sequential	`FnMut() -> Fut<Req>`

虽然 async-lsp 看似 async 了，但是别扭之处在于，它的 handler 无法使用 `.await` 语法，取而代之，必须返回一个不引用 state 的 async 闭包。这显然是 rust 的局限性。其次 async-lsp 将 stdio 的读写异步化了，而在 windows 上，这必须要借助 tokio 的 compat IO（correct me if I’m wrong，因为我不是 async 专家）。对于 nix，这并非问题，因为 nix 只会在 unix 上运行（correct me if I’m wrong，因为我不是 nix 用户）。我觉得 nil 的作者不会为 windows 用户买单属于合情合理。

另外，为了方便测试，我有一些希望 async-lsp 改变的东西（目前已经忘了）。出于以上原因，尽管 tinymist 已经完全做好了迁移到 async-lsp 的准备，我还是选择了继续保留对 lsp-server 的包装。总的来说，我希望将来要么 rust 改进对 async 借用的支持，要么有一个更好的引擎框架。