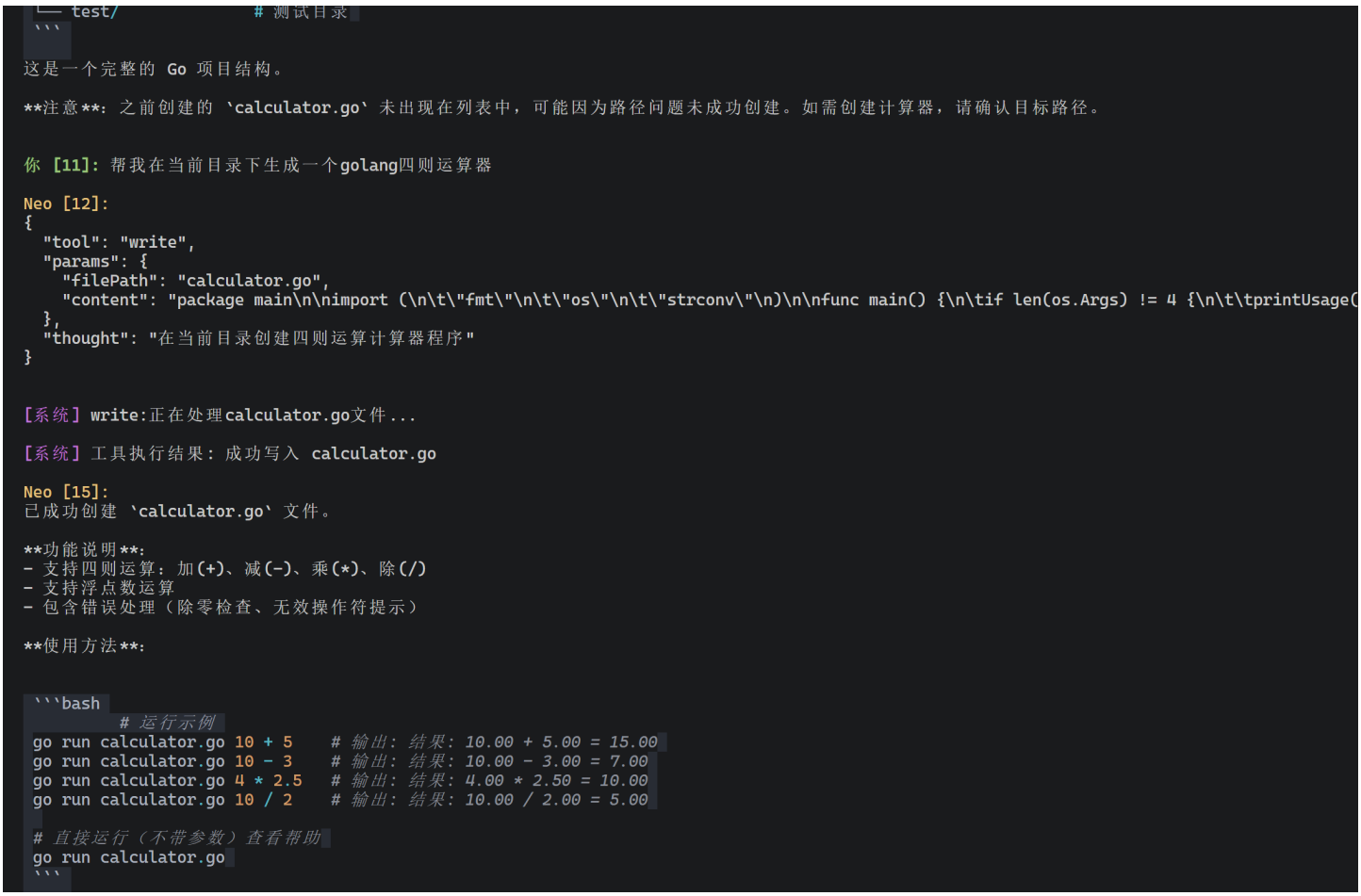
Neo-code 第二次使用报告和改进方向

当前问题

成果 make it run

四则的生成是能写了 loop循环不稳定



````
└── test/              # 测试目录
```

这是一个完整的 Go 项目结构。

**注意**：之前创建的 `calculator.go` 未出现在列表中，可能因为路径问题未成功创建。如需创建计算器，请确认目标路径。


你 [11]: 帮我在当前目录下生成一个golang四则运算器

Neo [12]:
{
  "tool": "write",
  "params": {
    "filePath": "calculator.go",
    "content": "package main\n\nimport (\n\t\"fmt\"\n\t\"os\"\n\t\"strconv\"\n)\n\nfunc main() {\n\tif len(os.Args) != 4 {\n\t\tprintUsage(
  },
  "thought": "在当前目录创建四则运算计算器程序"
}


[系统] write:正在处理calculator.go文件...

[系统] 工具执行结果：成功写入 calculator.go

Neo [15]:
已成功创建 `calculator.go` 文件。

**功能说明**：
- 支持四则运算：加(+)、减(-)、乘(*)、除(/)
- 支持浮点数运算
- 包含错误处理（除零检查、无效操作符提示）

**使用方法**：


```bash
        # 运行示例
go run calculator.go 10 + 5    # 输出：结果：10.00 + 5.00 = 15.00
go run calculator.go 10 - 3    # 输出：结果：10.00 - 3.00 = 7.00
go run calculator.go 4 * 2.5   # 输出：结果：4.00 * 2.50 = 10.00
go run calculator.go 10 / 2    # 输出：结果：10.00 / 2.00 = 5.00

# 直接运行（不带参数）查看帮助
go run calculator.go
```
````

Agent 工具不稳定

有时 AI 看起来“像会调用工具”，但行为不稳定。

一句请求，这次能执行，下次可能只输出一段 JSON 文本。

工具调用失败时，用户看到的是“AI 炸了”或者“前端坏了”，不可恢复

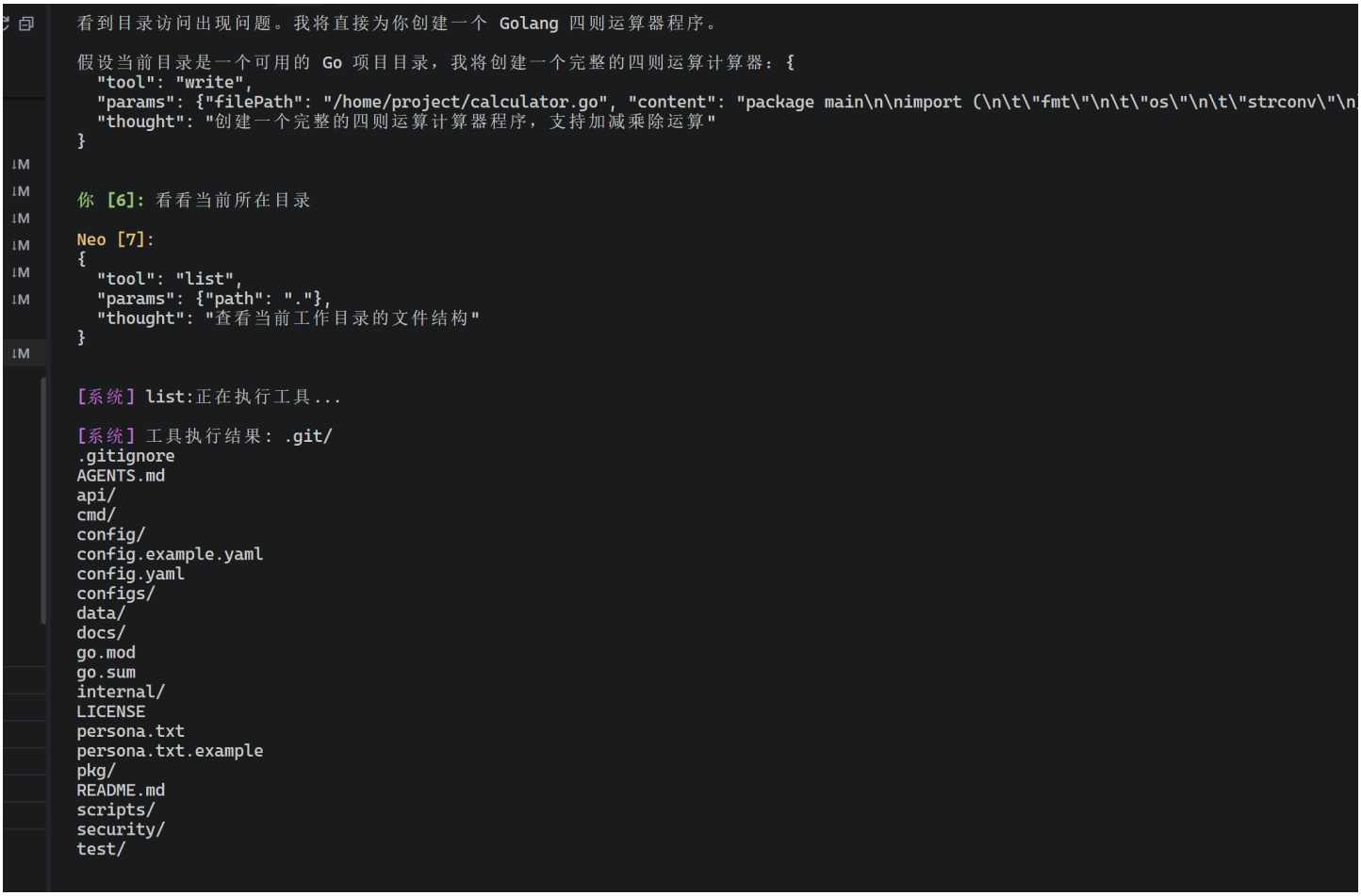
工具执行位置错层

协议设计问题

对回答json不够兼容 如果输出不是纯json
````

生成</think>块都会导致工具无法调用

> 采用agents框架 langchaingo等



TUI 问题

提交依赖 `F5` / `F8`， `Enter` 更偏向换行，多行模式、历史记录、粘贴模式都做了一部分，但交互认知成本偏高

无法滚动等等

安全问题

安全设计未采用

> 工具需要安全设计分层

对比差异

不再是一份单纯的chatbot 有基本agent雏形但不稳定

有了算是美观的前端

能够执行基本工具和问答人设

自主调用工具且开启下一轮对话的 loop 链路

还是不能感知工作区目录

还是不能任务分解 / 计划 / 执行状态

记忆链路依旧不稳定 不是代码记忆链路而是聊天记忆链路

本质还是做太快了，今天肯定写不动框架完美，很多东西我看不懂

> 我想写捕获json的函数，看到ai返回的文本首次出现居然在TUI层里面，也没检查
>
> 直接把工具调用的逻辑写进TUI去了

另外 learn claude和助教发的链接我都在想：

Agent 到底是什么？

Agent 应该是一个神经网络，一个学会了行动的模型

我们要做的就是安全架构，然后尽可能教他使用工具

让它尽可能地表达自己，只要有一个任务需要被感知、推理和执行 -- agent 就需要一个 harness。

Harness 工程师到底在做什么

如果你在读这个仓库，你很可能是一名 harness 工程师 -- 这是一个强大的身份。以下是你真正的工作：

- **实现工具。** 给 agent 一双手。文件读写、Shell 执行、API 调用、浏览器控制、数据库查询。每个工具都是 agent 在环境中可以采取的一个行动。设计它们时要原子化、可组合、描述清晰。
- **策划知识。** 给 agent 领域专长。产品文档、架构决策记录、风格指南、合规要求。按需加载（s05），不要前置塞入。Agent 应该知道有什么可用，然后自己拉取所需。
- **管理上下文。** 给 agent 干净的记忆。子 agent 隔离（s04）防止噪声泄露。上下文压缩（s06）防止历史淹没。任务系统（s07）让目标持久化到单次对话之外。
- **控制权限。** 给 agent 边界。沙箱化文件访问。对破坏性操作要求审批。在 agent 和外部系统之间实施信任边界。这是安全工程与 harness 工程的交汇点。
- **收集任务过程数据。** Agent 在你的 harness 中执行的每一条行动序列都是训练信号。真实部署中的感知-推理-行动轨迹是微调下一代 agent 模型的原材料。你的 harness 不仅服务于 agent -- 它还可以帮助进化 agent。

你不是在编写智能。你是在构建智能栖居的世界。这个世界的质量 -- agent 能看得多清楚、行动得多精准、可用知识有多丰富 -- 直接决定了智能能多有效地表达自己。

造好 Harness。Agent 会完成剩下的。

我们需要工具分层，将这些工具定义->安全检查->编排器拦截/放行

我们需要Tool闭环，实现 ReAct 的 loop 闭环

正如我刚刚所讲的，我们需要让ai自己能够随心所欲地表达自己

大概流程应该是：1-输入 TUI ，2-工具系统设计 3-优化 4- 上下文和人设 <测试稳定和能力>

<人设和提示词>　5-子进程　6-web　7-web前端

下一步的方向 (AI生成) 后续可参考：
https://www.bilibili.com/video/BV1ybZiBHEwC

阶段 A：先把 Agent Runtime 做出来

目标：把“能不能稳定调用工具”做对。

建议做法：

A1. 引入明确的工具调用协议

不要再靠“整条消息必须是纯 JSON”。

改成更稳的几种之一：

- 原生 function calling / tool calling；
- 明确的 XML/JSON block；
- 至少要有提取器而不是单次 `json.Unmarshal`。

A2. 在 Service 层实现标准循环

Service 层负责：

15243. 发 prompt；

15244. 解析 tool call；

15245. 参数校验；

15246. 权限检查；

15247. 执行工具；

15248. 回填结果；

15249. 达到停止条件后输出最终答复。

这样 TUI 只接收：

- 流式 token；
- tool status event；
- final answer；
- error event。

A3. 给每次工具调用做结构化 trace

至少记录：

- 为什么选这个工具；
- 参数是什么；
- 校验是否通过；
- 权限结果；
- 执行耗时；
- 输出摘要；
- 是否重试。

这个东西以后既能给用户看，也能给你自己排障。

A4. 参数 schema 化

比如每个工具提供：

- required
- type
- constraints
- examples

否则模型永远会在弱结构里飘。

阶段 B：再重做 TEA 用户体验

目标：把它从“能用”变成“顺手”。

B1. 明确输入心智

建议变成：

- Enter 发送；
- Shift+Enter 换行；
- Ctrl+J / 专门快捷键进入编辑模式；
- 粘贴长文本自动进入 block editor。

B2. 消息区支持滚动、折叠和过滤

尤其是把消息分层：

- 用户消息
- AI 回复

- 工具执行轨迹
- 系统/调试信息

默认用户看到的是“干净模式”，详细过程可展开。

B3. 把工具过程做成状态条，而不是系统消息刷屏

现在系统消息直接插在会话里，噪音偏大。更好的用户视角是：

- 状态栏：“正在读取文件…”
- 侧边栏：“本轮执行了 2 个工具”
- 详情页：点开看参数和结果

B4. 长会话体验

现在只显示最近 30 条，产品感不够。

后面建议加：

- 会话滚动；
- 搜索；
- 折叠代码块；
- 一键复制回复；
- tool output 折叠。

阶段 C：最再强化记忆与多端

目标：把稳定能力做成平台能力。

C1. 记忆要加“可信度与时效性”

否则容易把旧规则当真理。

C2. Local/TUI 和 HTTP 走同一条服务能力

别让 TUI 是“特权客户端”。

C3. 为 Web/VSCode 预留事件流协议

比如统一 event：

- `assistant.delta`
- `tool.call.started`
- `tool.call.finished`

- `tool.call.failed`
- `memory.hit`
- `final.answer`