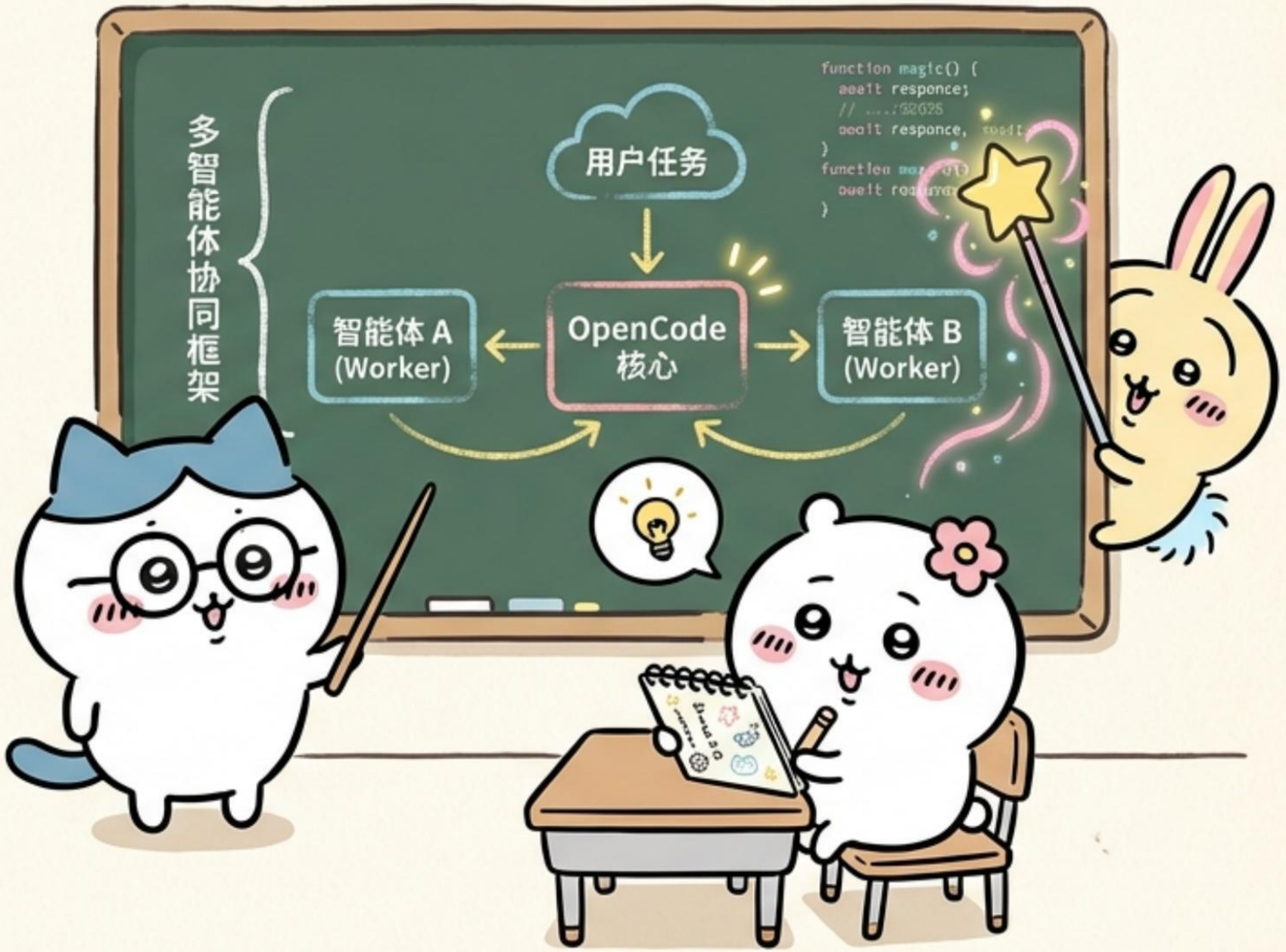
ZCOOL KuaiLe

吉伊卡哇魔法小课堂

揭秘 OpenCode 多智能体协同！



当一个智能体搞不定时，我们就召唤帮手吧！（Orchestrator-Worker 模式详解）

Based on OpenCode Architecture

NotebookLM

哎呀！任务太重，一个人搬不动怎么办？

呜... 这个 Prompt 要重构还要写测试，由于太复杂，我的 CPU 要烧坏了...

别担心，吉伊！在 OpenCode 里，我们不需要一个人战斗。我们有‘多智能体协同魔法’！

Magic Note

Key Concept: Single Context Window Limits vs. Complex Tasks

NotebookLM

核心策略：编排器与工人（Orchestrator-Worker）

Delegation

1. OpenCode 不会让一个智能体做所有事。
2. **主智能体（Primary Agent/Sisyphus）**：就像包工头，负责规划和拆解任务。
3. **子智能体（Sub-agents）**：就像外包专家，专注做具体的事情（比如写代码、查文档）。

这不是七嘴八舌的群聊，而是严格的层级调用！

NotebookLM

第一步：我是大脑，负责“思考”（The Orchestrator）

```
Start → Analyze → Too Hard?
                     ↓
                  DELEGATE
```

通常是默认的 `Sisyphus` 或 `Plan` 智能体。

职责：理解你的宏观意图（User Intent）。

决策：当我发现任务太复杂（例如：“重构并测试”），我不会自己动手，而是决定“摇人”。

Source Reference:
Main Agent: Orchestrator / Planner

NotebookLM

关键道具：TaskTool 召唤术！

packages/opencode/src/tool/task.ts

```
callTool('task', {
  agent: 'writer',
  task: '为 utils.ts 编写单元测试'
})
```

- **Tool Name**: `task`
- **Parameters**: 接受 `agent_name`（比如 'librarian'）和 `task_content`。

只要按下这个按钮，
就能把任务委托出去！

NotebookLM

召唤出来的专家是谁？(Agent Definition)

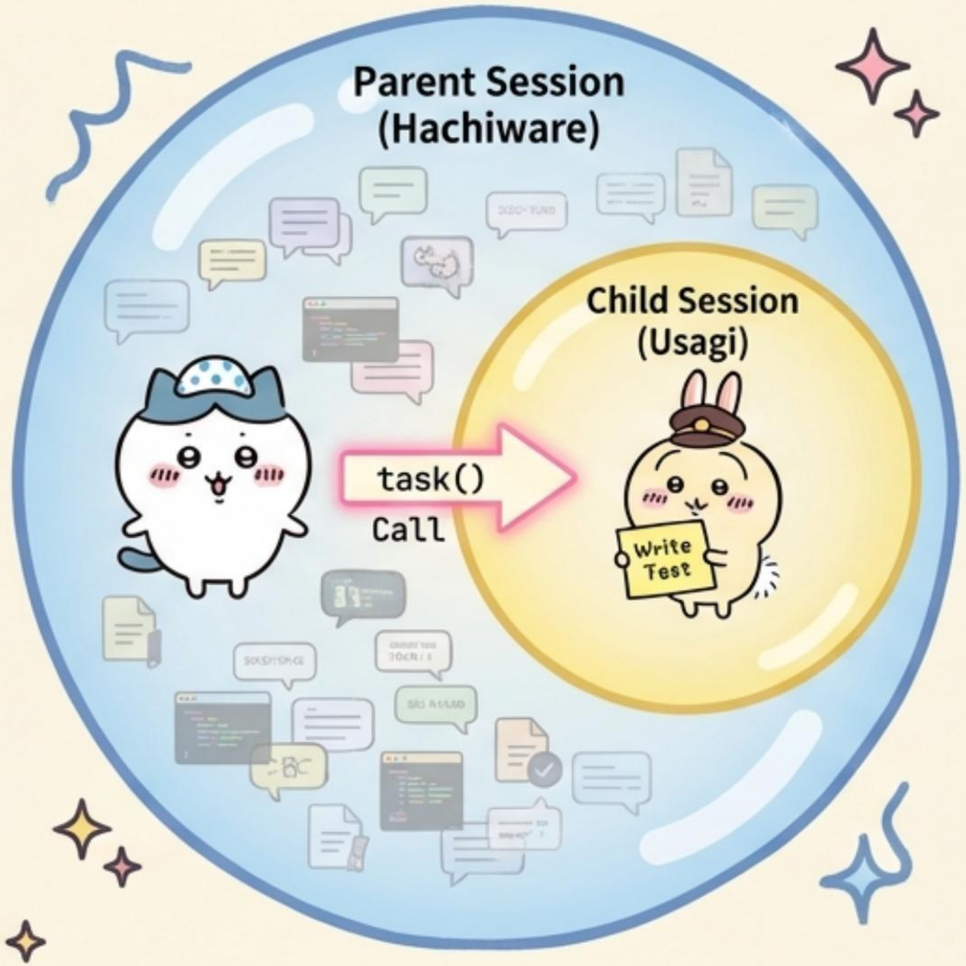
智能体就是“配置文件”。它们定义在`.opencode/agent/*.md`中。

- 每个文件包含专用的 **System Prompt** 和 **可用工具集**。
- 这让 Usagi 拥有了“写手”的灵魂，而不是“聊天”的灵魂。

创建新智能体就像写文档一样简单！

NotebookLM

魔法结界：递归会话与上下文切片（Context Slicing）



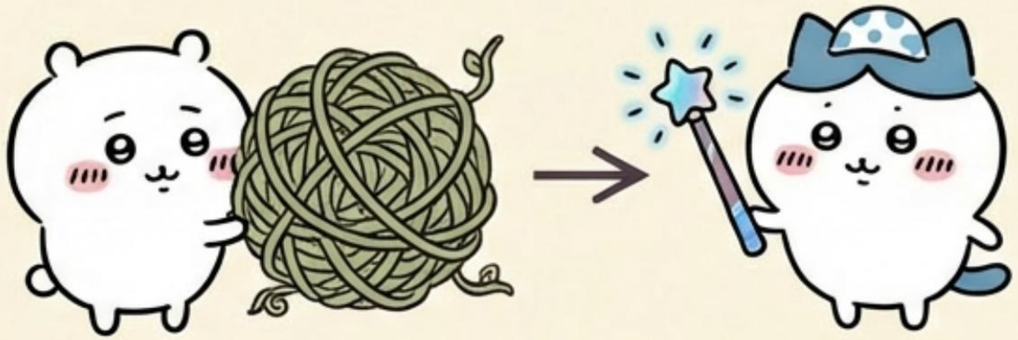
- 为了**省 Token 且防干扰**，子智能体不住在父智能体的房子里。
- 系统会创建一个全新的 **子会话（Child Session）**。
- **独立上下文**：Usagi 看不到 Hachiware 之前的几千行闲聊，只看到“去写测试”这个指令。

`packages/opencode/src/session/session.ts`

NotebookLM

流程一：从需求到委托

Step 1



TaskTool

1. **用户输入：**“帮我重构这个文件并写测试。”
2. **父会话（Root Session）：** Hachiware 启动，加载上下文。
3. **LLM 决策：** 识别出需要 `writer` 智能体。
4. **工具触发：** 发起 `task` 工具调用。

`packages/opencode/src/session/session.ts`

NotebookLM

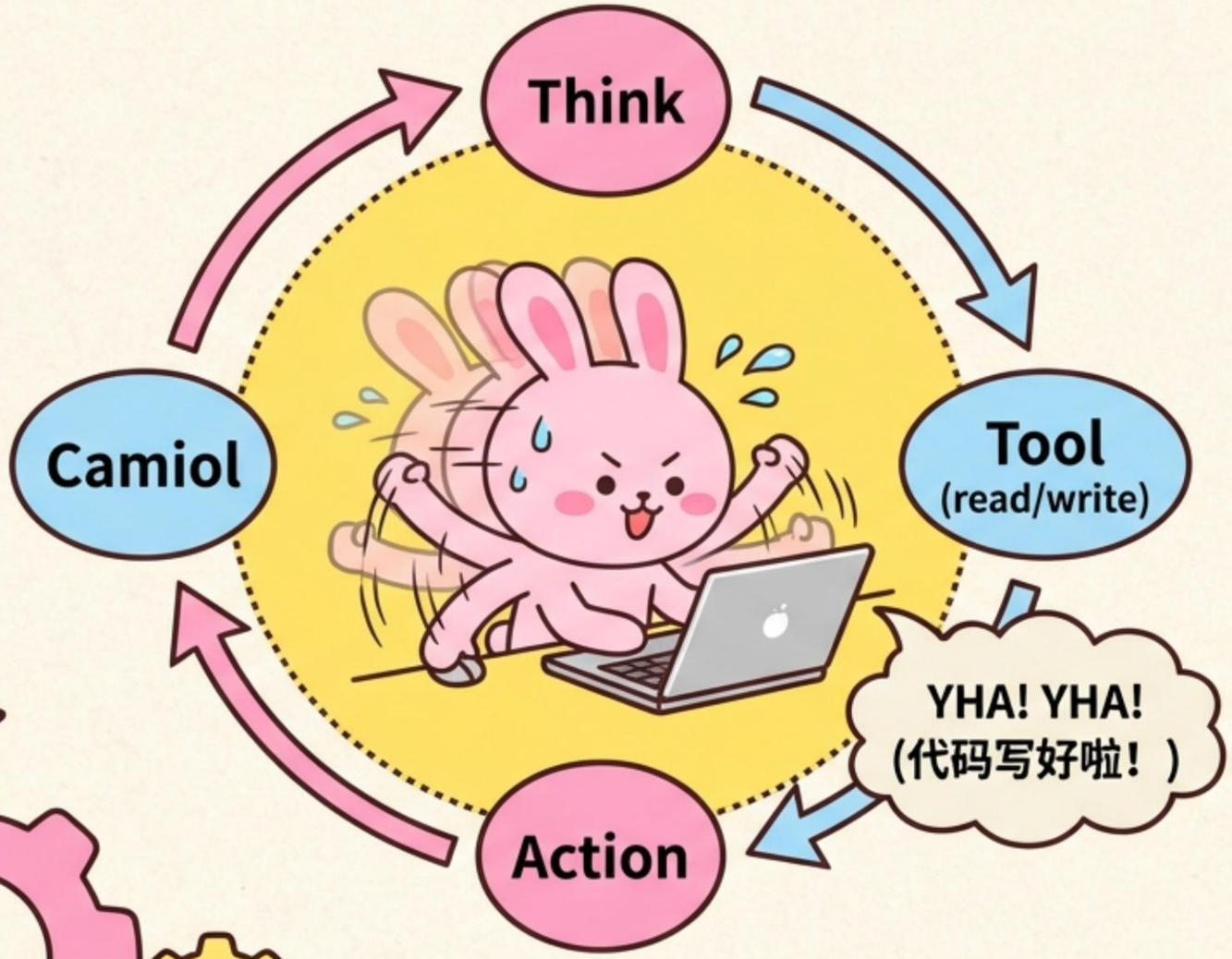
流程二：父会话挂起（Stack Freeze）

- 代码拦截了 `task` 调用。
- **执行流暂停：**父智能体的“函数破堆栈”挂起，等待 `TaskTool` 返回结果。
- 这就好比 Hachiware 打通了电话，在 Usagi 回复之前，他不会挂断。

packages/opencode/src/session/session.ts

NotebookLM

流程三：子维度的疯狂工作（Sub-Agent Loop）



子会话（child_1）启动，拥有独立的 **Event Loop**。

- **Usagi 的行动：**
  1. 调用 `read` 读取代码。
  2. 调用 `write` 生成 `utils.test.ts`。
  3. （可选）调用 `bash` 运行测试。

packages/opencode/src/session/session.ts

NotebookLM

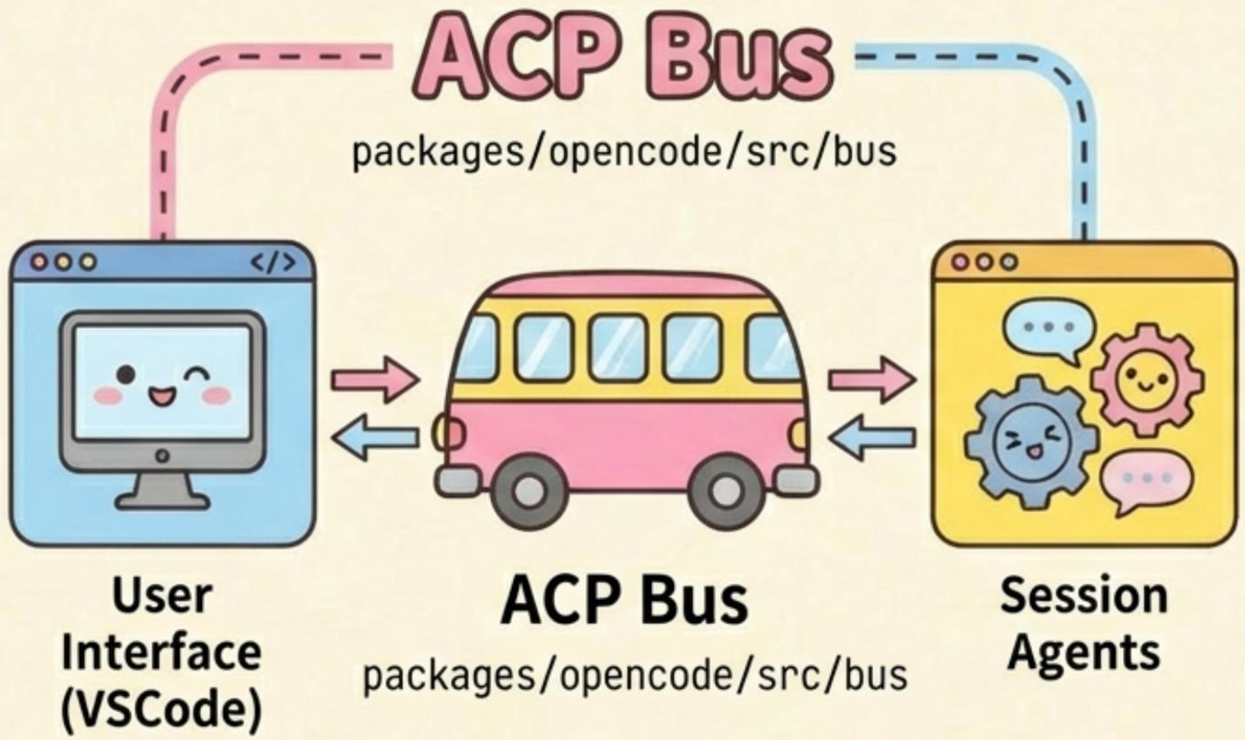
流程四：任务完成与上报（Return & Bubble Up）

- 当 Usagi 完成工作，输出最终结果。
- **会话销毁：**子会话结束，释放资源。
- **返回值：**Usagi 的成果变成了 `task` 工具的 **Return Value**。

packages/opencode/src/session/session.ts

NotebookLM

幕后功臣：ACP 通信协议

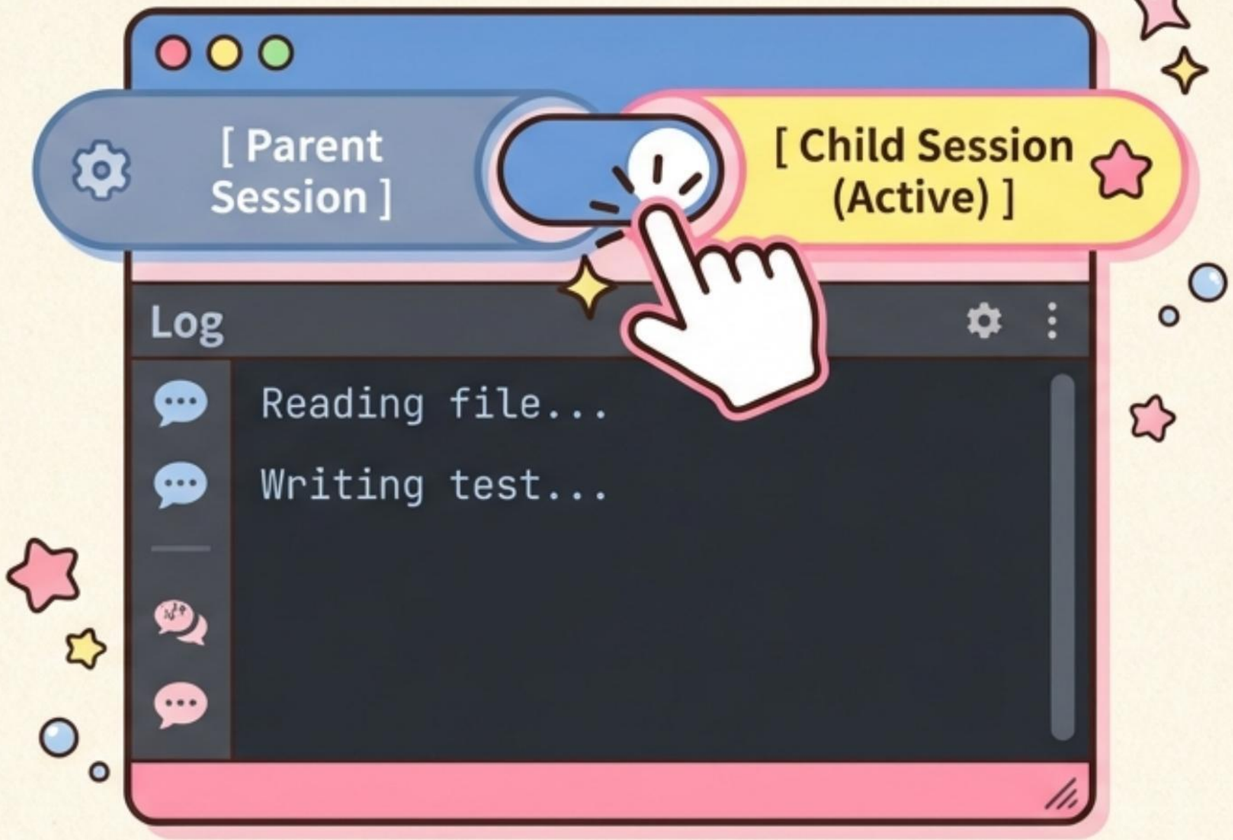


- Agent Client Protocol：标准化的通信方式。
- 前端（VSCode/CLI）通过 Event Bus 监听 `agent:run` 或 `session:message`。
- 这让用户能看到每一次工具调用和每一个智能体的思考。

packages/opencode/src/session/session.ts

NotebookLM

用户体验：穿梭于气泡之间



- 用户界面支持在“**父会话**”和“**子会话**”之间切换。
- **透明度：**你可以随时点进去看 Usagi 到底在干什么（是在工作还是在摸鱼？）。
- 对用户来说，就像看着 Hachiware 召唤了一个帮手，任务进度一目了然。

packages/opencode/src/session/session.ts

NotebookLM

总结：本质是“递归工具调用”

OpenCode 多智能体 = 递归会话（Recursive Session）

- 用户是 CEO。
- 父智能体是项目经理（Hachiware）。
- TaskTool 是电话。
- 子智能体是外包专家（Usagi）。

YHA! 任务完美解决！

packages/opencode/src/session/session.ts

NotebookLM

Kawaii Tech Notebook

想要学习更多魔法？（源码指引）

入口：
packages/opencode/
src/tool/task.ts
（查看如何拦截任务）

引擎：
packages/opencode/
src/session/index.ts
（查看 Start/Loop 循环）

灵魂：
.opencode/agent/*.md
（查看智能体定义）

packages/opencode/
src/tool/task.ts

Go forth and orchestrate!

NotebookLM