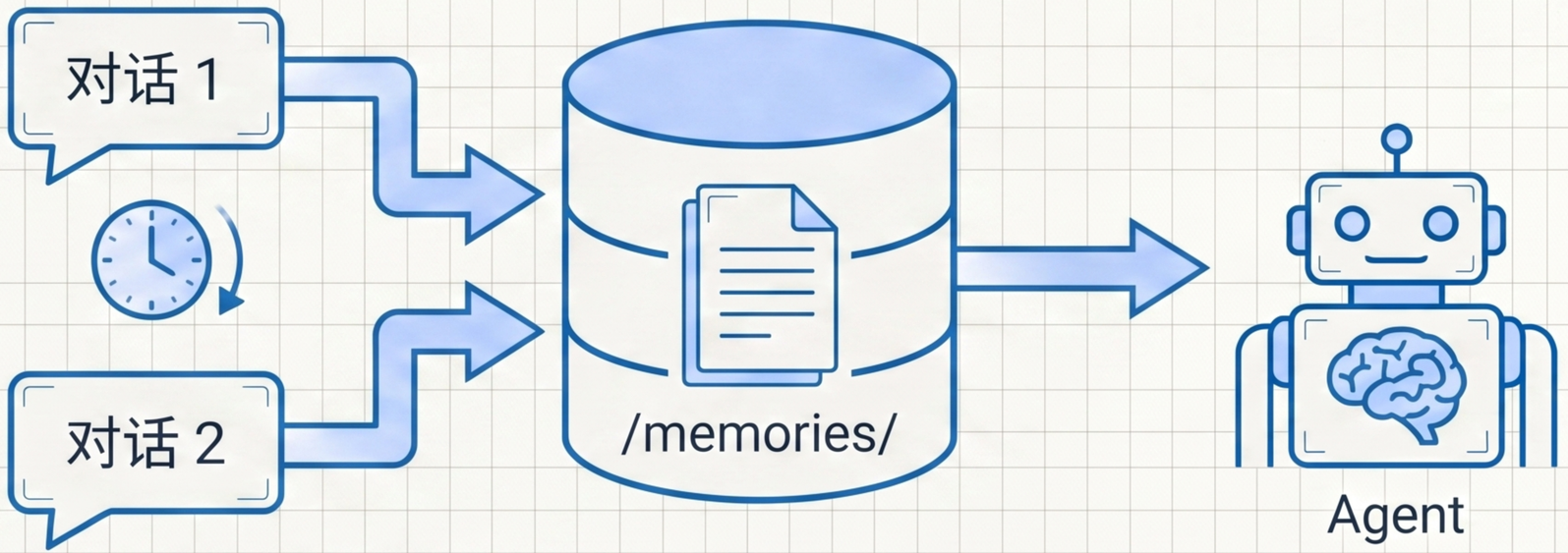
Deep Agents 实战

第 11 讲：长期记忆 — 让 Agent 拥有跨对话的记忆



短期（Checkpointer）　　长期（Store）　　路由（CompositeBackend）

Agent 的两种'记忆'

短期记忆 vs 长期记忆

短期记忆

- thread_id 内有效
- 对话结束消失
- Checkpointer

类比：工作桌面（下班清空）

长期记忆

- 跨 thread 持久化
- 下次对话可读取
- Store

类比：书架（永久保留）

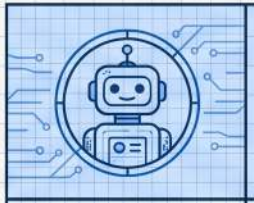
CompositeBackend 将两者组合

Memory 的工作原理

三步：声明 → 读取 → 更新

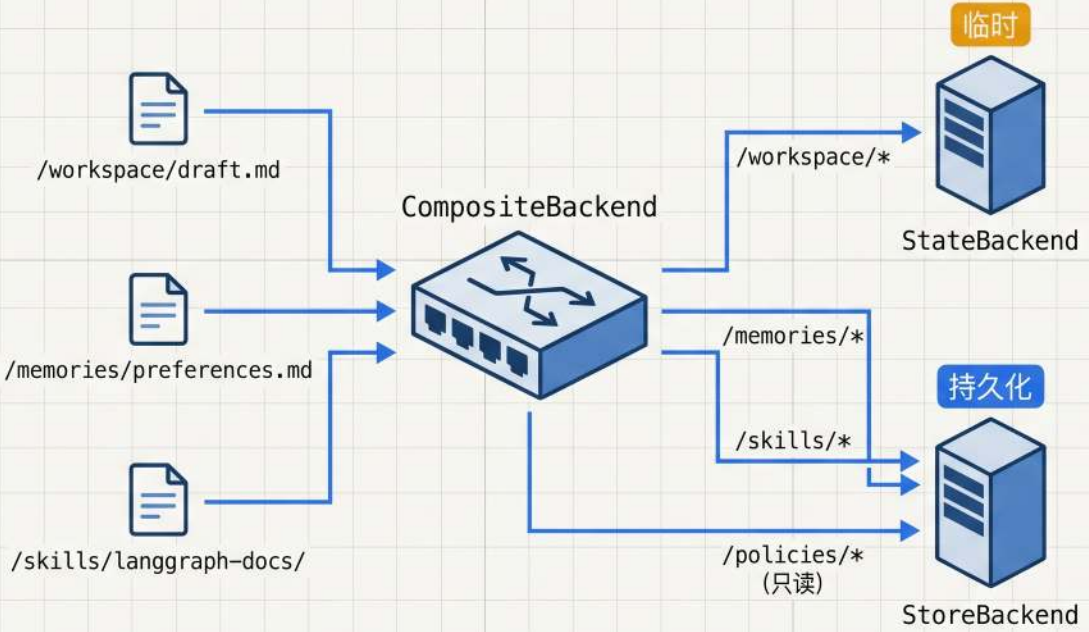
记忆的三种作用域

namespace 决定隔离粒度

	作用域	namespace	效果
	Agent 级	`(assistant_id,)`	所有用户共享，Agent 不断进化
	用户级	`(user_id,)`	按用户隔离，互不干扰
	组织级	`(org_id,)`	全组织共享，通常只读

CompositeBackend 路径路由

路径前缀决定存储后端



- /workspace/* → StateBackend（临时，对话结束丢失）
- /memories/* → StoreBackend（持久化，跨对话保留）
- /skills/* → StoreBackend（程序性记忆）
- /policies/* → StoreBackend（组织策略，只读）
- Agent 使用同样的 write_file / read_file 工具，路由对 Agent 透明

跨对话访问

不同 thread_id 共享持久化文件

核心：memory= 参数让 Agent 启动时自动加载

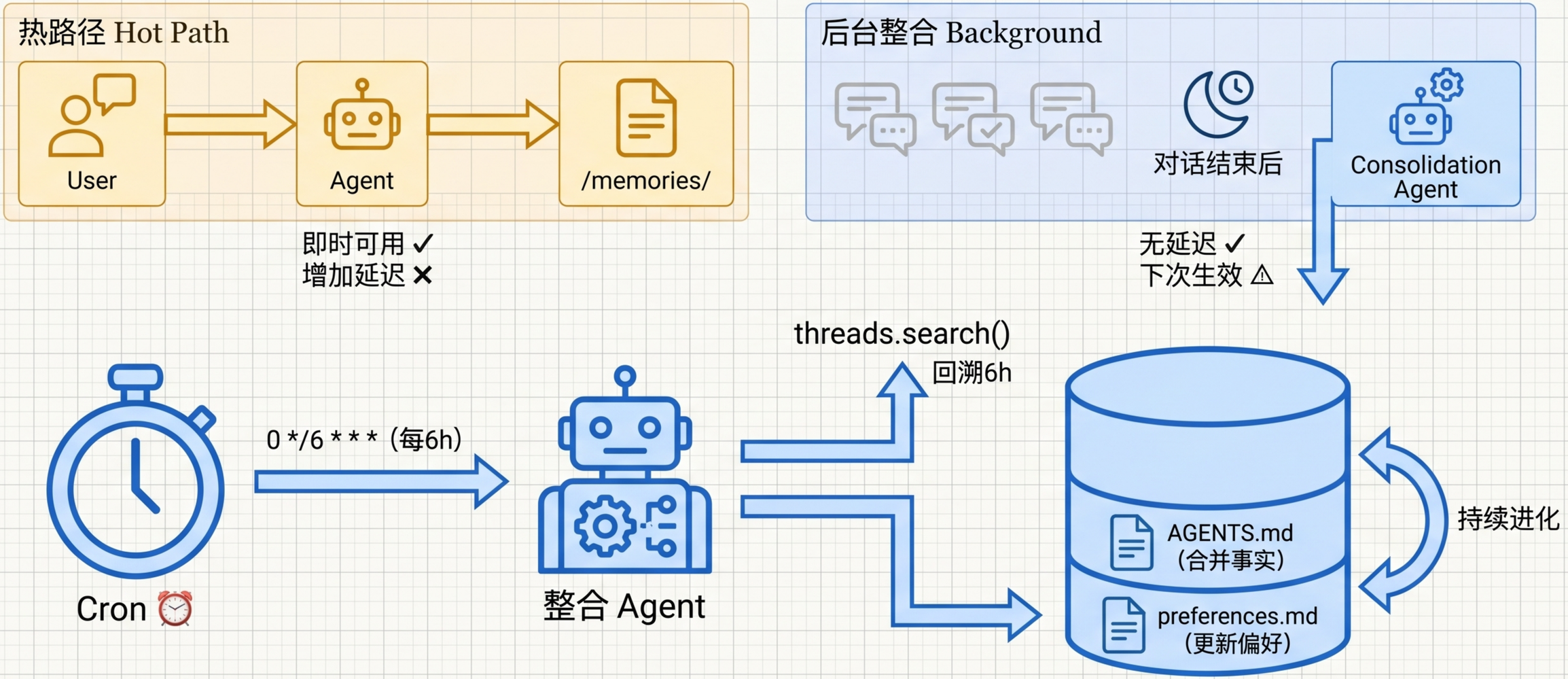
记忆的六个维度

每个维度独立可配置

	持续时间	短期 / 长期
	信息类型	情景 / 程序性(Skills) / 语义
	作用域	用户 / Agent / 组织
	更新策略	对话中 / 对话间(后台整合)
	检索方式	启动加载 / 按需读取
	权限控制	读写 / 只读

后台记忆整合

Sleep-time Compute — 对话间异步处理



从开发到生产

Store 升级路径

开发

InMemoryStore

InMemoryStore

- 零配置，重启丢失

生产

PostgresStore

PostgresStore

- 真正持久化，可伸缩

托管

LangSmith

LangSmith

- 平台自动提供 Store — 无需手动配置

store.put() + create_file_data() 初始化记忆

本讲回顾

核心

- Memory 一等公民：memory= 参数声明，启动时自动加载
- 三种作用域：Agent 级 / 用户级 / 组织级，namespace 控制隔离
- CompositeBackend：路径前缀路由到不同后端，对 Agent 透明
- 高级特性：情景记忆搜索、后台整合 Cron、读写权限控制
- 升级路径：InMemoryStore → PostgresStore → LangSmith 托管
- 下一讲：Human-in-the-Loop — 为敏感操作添加人工审批

下一讲：**Human-in-the-Loop**
为敏感操作添加人工审批