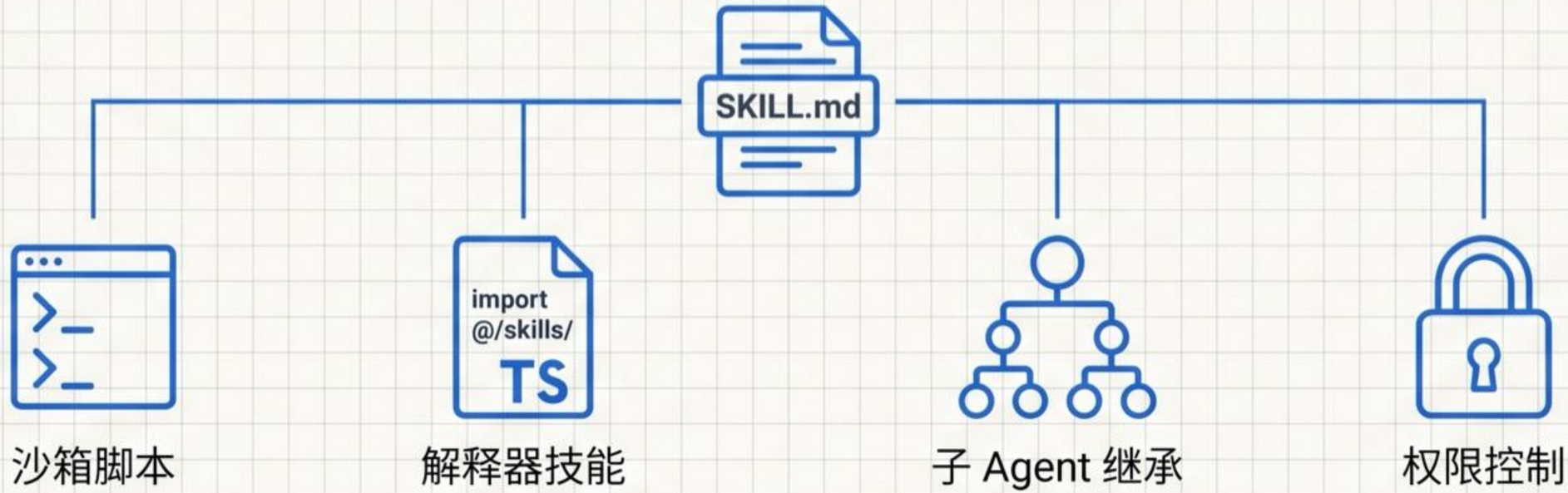
Deep Agents 实战

第 10 讲:Skills(下) — 高级用法

沙箱脚本、解释器技能、子 Agent 继承与权限控制



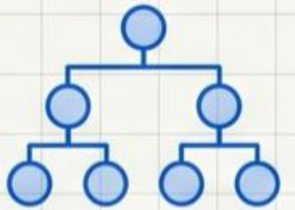
本讲地图

从“会用 Skills”到“用好 Skills”

代码执行

用 Skills 执行代码:沙箱脚本 + 解释器技能

子 Agent



Skills 与子 Agent:继承规则与状态隔离

权限控制

Skill 权限控制:deny / interrupt / 共享分层

三者对比

Skills、Memory 与 Tools:三者如何分工

用 Skills 执行代码

两种模式，两种适用场景

SKILL.md 中的可执行代码

沙箱脚本

隔离容器执行

`scripts/search.py`

- scripts/ 下的脚本在隔离容器中运行
- 适合 Shell、安装包、文件系统访问

解释器技能

确定性函数调用

`import @/skills/`

- Agent import 经过测试的函数
- 适合确定性的 JS/TS helper

读取脚本：任何后端都可以；执行脚本：需要对应的运行环境

沙箱脚本:在隔离环境中跑代码

SKILL.md 描述意图,scripts/ 提供实现

skills/arxiv-search/

- SKILL.md
- scripts/search.py

读取说明 → 容器内执行

- 目录结构:SKILL.md(说明) + scripts/(可执行脚本)
- Agent 读取 SKILL.md，按 Instructions 调用脚本
- 脚本在 Daytona 等沙箱容器中运行，与主进程隔离
- 容器内可自由装包、读写文件、执行 Shell

沙箱脚本示例:arxiv-search

SKILL.md = 元数据 + 说明 + 脚本调用

````
---
name: arxiv-search
description: 在 arXiv 上搜索学术论文并返回结构化结果
---

# arXiv 论文搜索

## Instructions

当用户需要查找学术论文时，运行以下脚本：

```bash
python scripts/search.py "<查询关键词>" --max-results 10
```

脚本会调用 arXiv API，返回标题、作者、摘要与链接。
````

元数据：被注入系统提示

自然语言说明：何时用

脚本调用：容器内执行

解释器技能：import 而非现写

把确定性逻辑沉淀成可复用函数

现写代码（问题）

问题：让 Agent 每次现写代码，结果不稳定、易出错

---

import 技能函数（方案）

方案：把经过测试的函数放进技能，Agent 直接 import 调用

- CodeInterpreterMiddleware 在 QuickJS 中执行，沙箱安全
- 适合确定性的数据处理：分组、排序、格式化等

解释器技能示例:order-helpers

元数据声明入口 + 函数实现 + 中间件挂载

1 声明 entrypoint (yaml / SKILL.md frontmatter)：

```
---
name: order-helpers
description: 订单数据处理工具函数
metadata:
  entrypoint: scripts/index.ts
---
```

2 导出可被 import 的函数 (typescript / scripts/index.ts)：

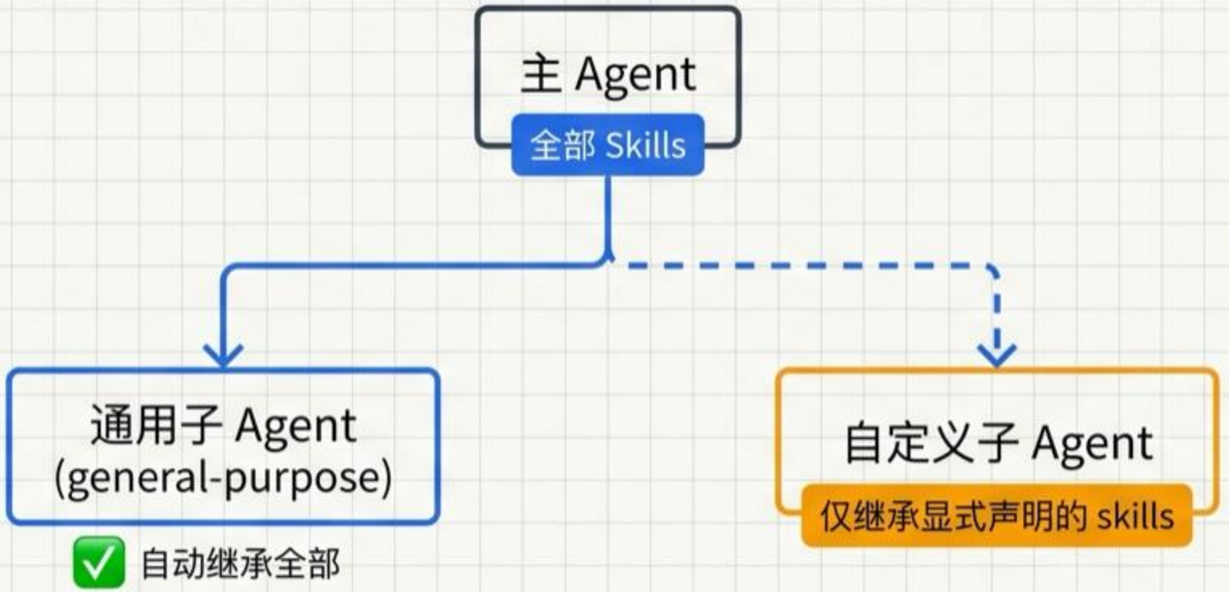
```
export function groupByStatus(orders: Order[]) {
  return orders.reduce((acc, o) => {
    (acc[o.status] ??= []).push(o)
    return acc
}, {} as Record<string, Order[]>)
}
```

3 中间件挂载技能后端 (python / 接入)：

```
agent = create_deep_agent(
    middleware=[CodeInterpreterMiddleware(skills_backend=backend)],
)
```

Skills 与子 Agent:谁继承?

通用继承，自定义显式声明



通用子 Agent(general-purpose): 自动继承主 Agent 的全部 Skills
自定义子 Agent: 默认不继承, 需在 skills 字段显式列出
状态隔离: 子 Agent 拥有独立的文件系统视图
设计意图: 让子 Agent 的能力边界清晰可控

子 Agent × Skills 示例

自定义子 Agent 显式声明 skills 字段

```
research_subagent = {
    "name": "researcher",
    "description": "负责深度调研的子 Agent",
    "tools": [web_search],
    "skills": ["/skills/research/", "/skills/web-search/"],
}

agent = create_deep_agent(
    skills=["/skills/main/"],
    subagents=[research_subagent],
)
```

显式声明:自定义子 Agent 才能用

工具与技能分别配置

主 Agent 的技能,通用子 Agent 自动继承

Skill 权限控制

三级策略，守住敏感操作

deny — 直接拒绝

Agent 无法对匹配路径执行操作

interrupt — 人工确认

需配 checkpointer

allow — 默认放行

默认放行

按 operations + paths 匹配

用 FilesystemPermission 精确匹配

权限控制示例:deny 与 interrupt

同一 API,两种保护强度

deny:写 /skills/ 直接报错

```
FilesystemPermission(
    operations=["write"],
    paths=["/skills/**"],
    mode="deny",
)
```

interrupt:写 /skills/ 暂停等确认

```
FilesystemPermission(
    operations=["write"],
    paths=["/skills/**"],
    mode="interrupt",
)
# interrupt 需配 checkpointer
agent = create_deep_agent(
    checkpointer=MemorySaver(),
)
```

interrupt 必须配 checkpointer

共享技能 + 个人技能

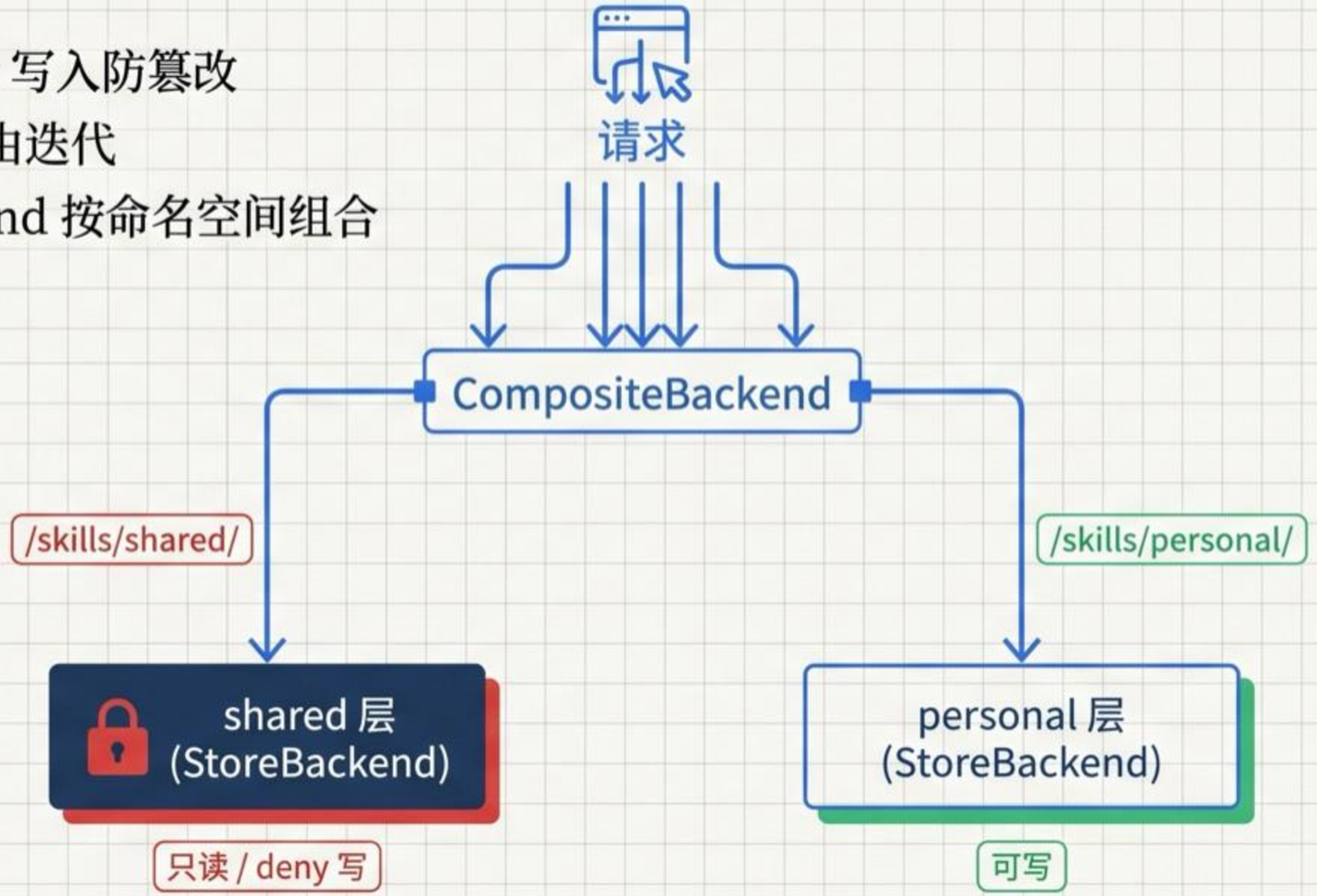
CompositeBackend 分层组合

共享层(/skills/shared/): 团队只读，deny 写入防篡改

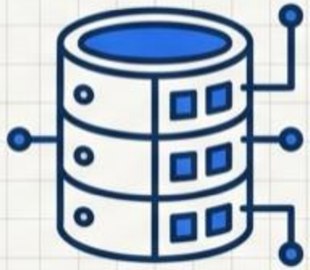
个人层(/skills/personal/): 用户可写，自由迭代

CompositeBackend 把多个 StoreBackend 按命名空间组合

请求按路径路由到对应后端



Skills、Memory 与 Tools
三者分工,各司其职

Skills	Memory	Tools
		
可复用的能力包(说明 + 脚本 + 函数)	跨会话的持久状态与上下文	单次原子操作的接口

本讲回顾： 沙箱脚本 · 解释器技能 · 子 Agent 继承 · 权限控制 · 分层共享

下一讲：长期记忆(Memory)