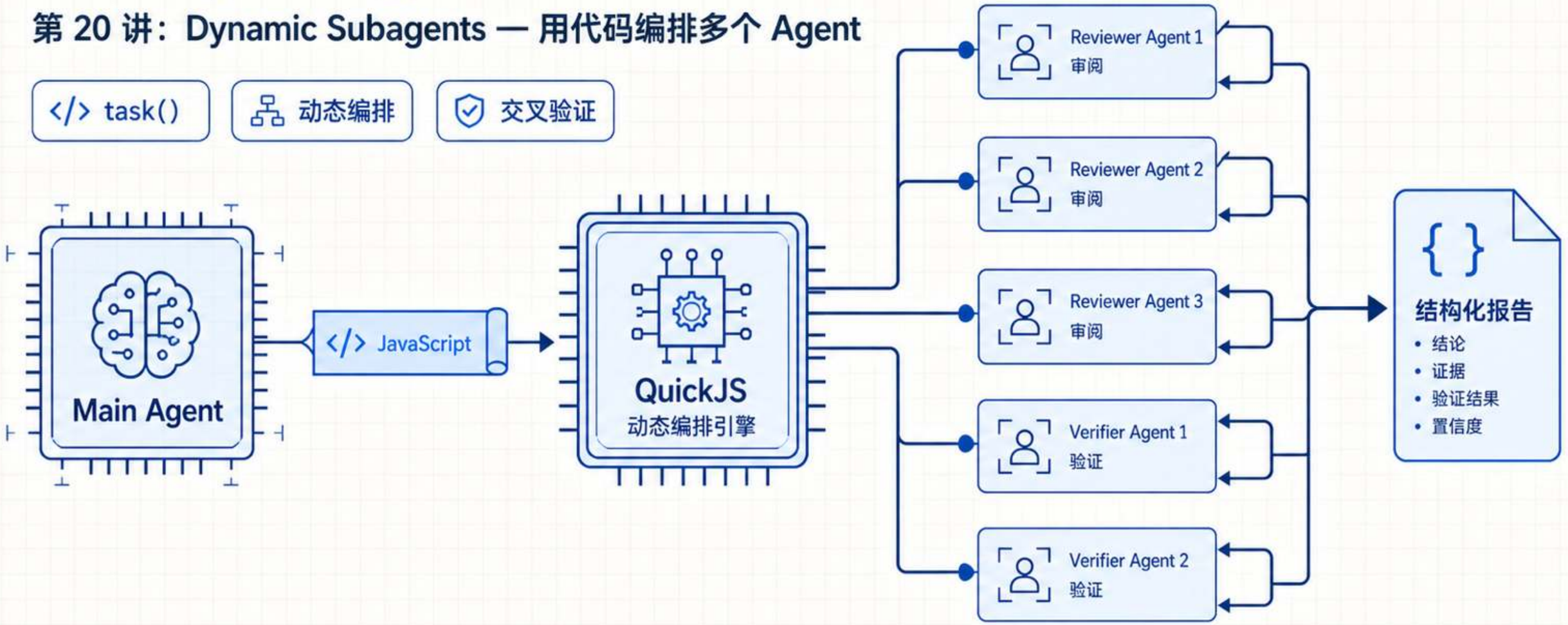
Deep Agents 实战

第 20 讲：Dynamic Subagents — 用代码编排多个 Agent

`</> task()` 动态编排 交叉验证

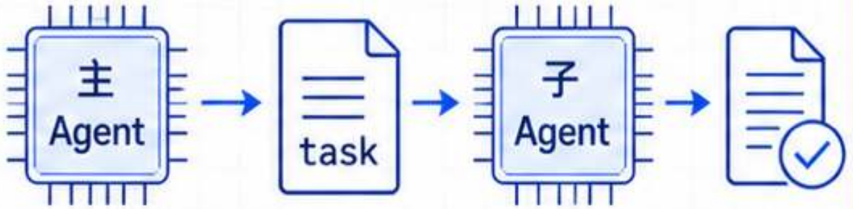


单次委派，还是代码编排？

一次明确交接用普通 task Tool；批量或多阶段流程再考虑代码编排

普通 task Tool

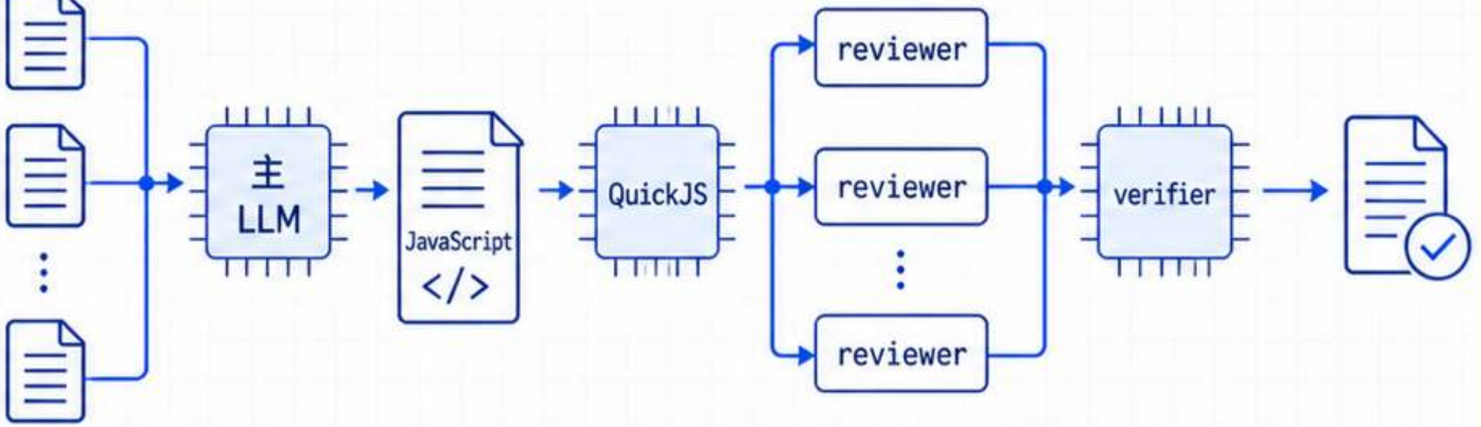
一个明确任务 → 一个子 Agent



代码编排

完整输入 → 主 LLM → JavaScript → QuickJS

reviewer 覆盖输入 → verifier 独立复核



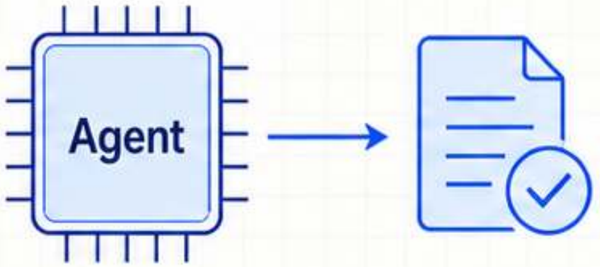
⚠ 生成代码可以表达覆盖规则与阶段关系，但不保证每次调度或判断都正确

先看任务形状，再选择子 Agent 机制

单次委派、批量编排和后台生命周期不是一回事

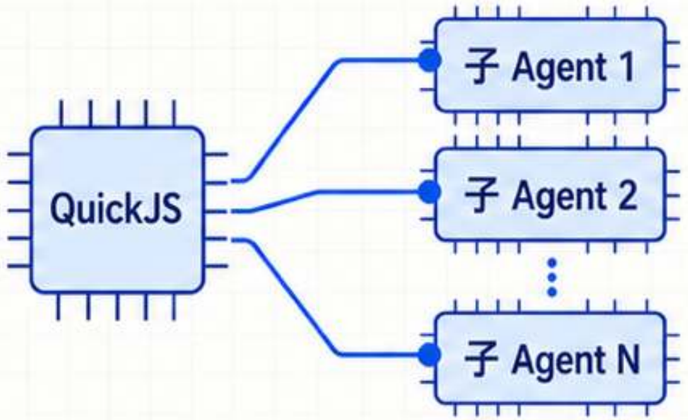
任务怎样展开？

单个明确任务 → 普通 task Tool



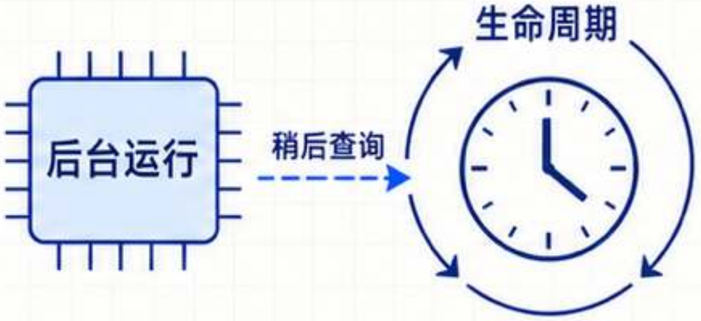
专业 Agent（单一专长）

批量 / 多阶段 → Dynamic Subagents



并行 / 条件 / 顺序编排

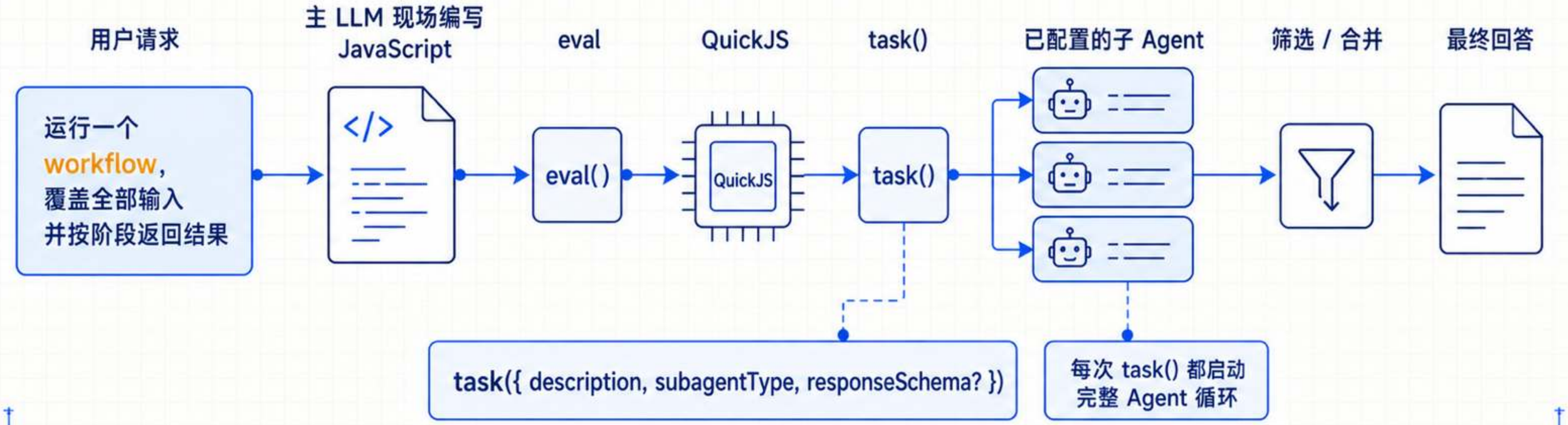
后台持续运行 → Async Subagents



异步执行 + 生命周期管理

JavaScript 是怎样跑起来的

workflow 提醒主 LLM 考虑编排，能力来自 subagents 与 CodeInterpreterMiddleware



workflow 是提示词关键词，不是配置项

task() 只有三个字段，也会启动完整 Agent 循环

两个必填字段选择任务与角色，一个可选 Schema 让结果可组合

description · 必填 必填

目标、范围、约束、输出要求必须自包含

subagentType · 必填 必填

必须匹配 subagents 中的 name

responseSchema · 可选 可选

设置后直接得到 JavaScript 对象

task()

```
task({
  description: "读取 src/auth.ts，检查
    认证绕过；返回行号与证据",
  subagentType: "reviewer",
  responseSchema: findingsSchema
})
```

读取上下文 → 调用工具 → 继续推理 → 返回结果

有 Schema → JavaScript 对象

无 Schema → 本次运行结果

findingsSchema：自定义 Schema，完整代码需先定义

一次调用返回一次运行结果，不是可继续对话的会话

先准备角色，再描述流程

配置好的角色决定“能调度谁”，流程提示决定“怎样调度”

开发者先准备

subagents： name / description / system_prompt

CodeInterpreterMiddleware(subagents=True)

工单：classifier / bug-fixer / feature-analyst / support-agent

审查：reviewer / verifier

死代码：analyzer

用户再描述

- workflow
- 输入范围
- 角色顺序
- 结果要求
- 停止条件

主 LLM

为当前任务生成 JavaScript

角色配置 + 流程提示 → 一次具体编排

用 AgentSeek Template 跑通 Dynamic Subagents

deepagents/subagents-dynamic 把六种编排场景、运行证据与生命周期一起打包

安装与创建

01 安装 AgentSeek

```
uv tool install agentseek
```

02 创建应用模板

```
agentseek create deepagents/subagents-dynamic --checkout main --no-input
```

```
cd dynamic_subagents_lab
```

配置与启动

03 准备环境

```
cp .env.example .env
cp frontend/.env.example frontend/.env
编辑 .env：模型、Base URL 与 API Key
```

04 安装依赖并启动

```
agentseek task sync
agentseek task frontend
agentseek dev
```

启动后得到

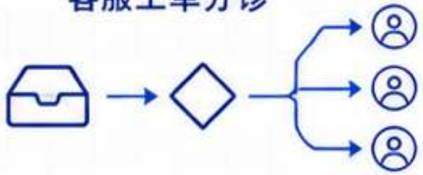
需要可调用 Tool 的模型；先在 .env 配好凭据，再运行场景

一个模板，六种可观察的编排场景

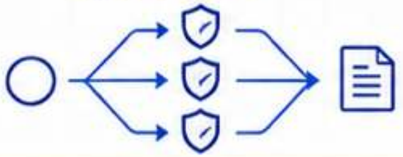
每个场景使用独立 assistant ； 页面只在真实运行证据出现后点亮

选择验证场景

1 Classify and act · 客服工单分诊



2 Fan-out and synthesize · 路由安全检查

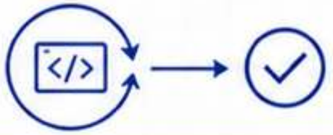


3 Adversarial verification · 支付漏洞复核

4 Generate and filter · 订单模型优选

5 Tournament · 可读性重构赛

6 Loop until done · 死代码收敛



运行证据

1 Workflow keyword
检测请求中的 workflow

2 Dynamic triggered
观察到解释器 tool call

3 Agents dispatched
来自真实 subagent stream

4 Workflow complete
解释器成功 + 子 Agent 完成 + 最终回答

可展开的运行证据

{ } Generated QuickJS

>_ Interpreter result

只有 workflow 关键词，不代表已经触发 Dynamic Subagents

场景一：先分类，再让受控映射选择处理角色

每条工单只进一个分支；unknown 保留未处理原因，不要猜

已配置的子智能体

- name：classifier
  description：工单分类
  system_prompt：不确定返回 unknown
- name：bug-fixer
  description：调查缺陷
  system_prompt：返回复现步骤与影响
- name：feature-analyst
  description：评估功能
  system_prompt：说明价值、条件与取舍
- name：support-agent
  description：回答使用问题
  system_prompt：无依据时明确说明

建议的 workflow 提示词

运行一个 workflow，处理：

T-101 登录后返回 500；

T-102 订单页导出 CSV；

T-103 如何重置密码。

先让 classifier 分类，再交给对应角色。

每条只进入一个分支；unknown 不猜。

按 ID 返回类别、结果和未处理原因。

可能生成的 JavaScript / 一种可能写法

```
async function route(t) {
  const kind = await task({ description: `分类 ${t.id}; ${t.text}; 仅返回受控类别`,
    subagentType: "classifier", responseSchema: categorySchema });
  const ROLE = { bug: "bug-fixer", feature: "feature-analyst",
    question: "support-agent" };
  const handler = ROLE[kind.category];
  if (!handler) return { id: t.id, category: "unknown", reason: "未处理" };
  return task({ description: `处理 ${t.id}: ${t.text}; 返回结论与依据`,
    subagentType: handler });
}
await Promise.all(tickets.map(route));
```

bug → bug-fixer · feature → feature-analyst · question → support-agent · unknown → 未处理

categorySchema：自定义 enum Schema，完整代码需先定义

验收：三个 ID 都出现 · 类别受约束 · unknown 不被强行路由

场景二：三个文件并行审查，再独立复核

reviewer 先覆盖输入，verifier 再确认或反驳；最多复核 20 条

已配置的子智能体

name: reviewer

description: 审查代码并给证据

system_prompt: 只报告有代码证据的候选

name: verifier

description: 独立复核候选

system_prompt: 先找反证再确认或反驳

建议的 workflow 提示词

运行一个 workflow，审查：

src/routes/login.ts

src/routes/session.ts

src/routes/reset-password.ts

reviewer 返回稳定 ID、文件、行号和证据；

按稳定 ID 去重；verifier 最多复核 20 条。

返回 confirmed，并给出 rejected、unreviewed 数量。

可能生成的 JavaScript / 一种可能写法

```
const files = ["src/routes/login.ts", "src/routes/session.ts",
  "src/routes/reset-password.ts"];
const reviews = await Promise.all(files.map(file => task({
  description: `审查 ${file}; 返回稳定 ID、文件、行号和证据`,
  subagentType: "reviewer", responseSchema: findingsSchema
})));
const unique = [...new Map(reviews.flatMap(r => r.findings)
  .map(x => [x.id, x])).values()];
const sample = unique.slice(0, 20);
const verdicts = await Promies.all(sample.map(x => task({
  description: `复核 ${x.id} @ ${x.file};${x.line}; 证据：${x.evidence}; 先找反证`,
  subagentType: "verifier", responseSchema: verdictSchema
})));
const confirmed = verdicts.filter(v => v.status === "confirmed");
const rejected = verdicts.filter(v => v.status === "rejected");
({ confirmed, rejected, unreviewed: unique.length - sample.length });
```

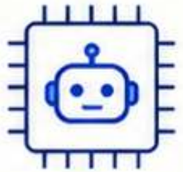
3 个文件 → reviewer → 稳定 ID 去重 → 前 20 条 → verifier → confirmed / rejected / unreviewed

findingsSchema / verdictSchema：自定义对象 Schema，完整代码需先定义

场景三：本轮没有新增就停止

在 src/legacy/ 中最多运行 5 轮，每轮最多保留 20 项

已配置的子智能体



name: analyzer

description: 分轮查找死代码

system_prompt: 使用稳定 ID，不重复已发现项

建议的 workflow 提示词

运行一个 workflow，在 src/legacy/ 中分轮查找死代码。

把已发现的稳定 ID 传给 analyzer，避免重复；

本轮没有新增就停止；最多 5 轮，每轮最多保留 20 项。

返回去重结果、实际轮数和停止原因。

可能生成的 JavaScript / 一种可能写法

```
const seen = new Set(), found = [];
let rounds = 0, stopReason = "最多 5 轮";
for (let i = 0; i < 5; i++) {
  const { items } = await task({
    description: `继续查找 src/legacy/ 的死代码；排除 ${[...seen]};
      每项返回稳定 ID、文件和证据；最多 20 项`,
    subagentType: "analyzer", responseSchema: itemsSchema });
  const fresh = items.filter(x => !seen.has(x.id)).slice(0, 20);
  rounds++;
  if (!fresh.length) { stopReason = "本轮无新增"; break; }
  fresh.forEach(x => seen.add(x.id)); found.push(...fresh);
}
({ found, rounds, stopReason });
```

itemsSchema：自定义对象 Schema，完整代码需先定义

5 轮 / 20 项来自业务提示，不是中间件参数

Dynamic Subagents 仍受同一组 Interpreter 参数约束

资源、状态、能力与调试设置继续约束动态编排

一组偏保守的起点

```
CodeInterpreterMiddleware(
  mode="turn",
  subagents=True,
  memory_limit=64 * 1024 * 1024,
  timeout=30.0,
  capture_console=True,
  max_result_chars=8000,
  ptc=["glob"],
  max_ptc_calls=64,
  max_snapshot_bytes=64 * 1024 * 1024,
)
```

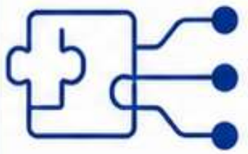
资源

memory_limit ·
timeout ·
max_result_chars

状态

mode ·
max_snapshot_bytes

能力



subagents ·
ptc ·
max_ptc_calls

调试与命名

capture_console ·
tool_name

max_ptc_calls 只限制 tools.*，不限制 task()；
当前没有 max_task_calls

动态调度数量仍要靠输入批次、循环上限、
Prompt 约束和业务预算控制

代码可以不同，流程语义要对

检查约束是否落实，不要求生成代码逐字一致

01 角色存在

subagentType 来自已配置的 name

02 字段正确

description · subagentType · responseSchema?

主 LLM 生成的 JavaScript

03 流程一致

输入逐项覆盖 · 阶段顺序符合提示

04 边界生效

数量上限与停止条件实际进入代码

逐项核对角色、字段、输入覆盖、阶段顺序与停止条件