OpenAI Agents 常用代理模式

本目录包含了使用 OpenAI Agents Python SDK 实现的多种代理模式示例。这些模式展示了如何构建和组织 AI 代理系统来解决各种复杂问题。

目录

1. 确定性流程

确定性流程是一种将复杂任务分解为一系列较小步骤的模式。每个步骤由一个专门的代理执行，前一个代理的输出作为下一个代理的输入。

主要特点：

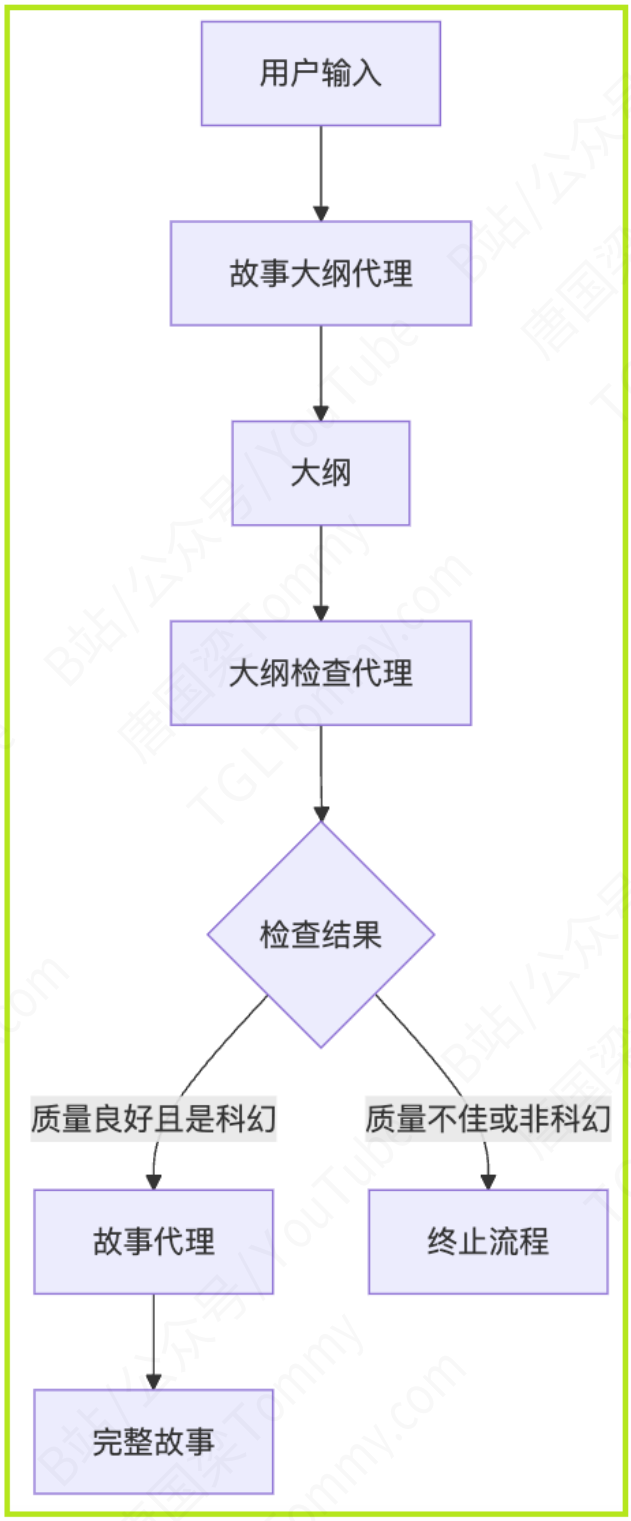
- 通过多个代理构建处理流水线
- 前一个代理的输出作为下一个代理的输入
- 在步骤之间可以添加决策点
- 使用 Pydantic 模型定义结构化输出

示例实现： `deterministic.py` 展示了一个故事生成流程：

1. 第一个代理生成故事大纲
2. 第二个代理检查大纲质量和类型（是否为科幻故事）
3. 如果质量不佳或不是科幻故事，流程终止
4. 如果质量良好且是科幻故事，第三个代理根据大纲写出完整故事

适用场景：任何可以明确拆分为顺序步骤的任务，如内容创作、数据处理流程等。

流程图：



2. 交接与路由

在许多情况下，您可能有专门处理特定任务的子代理。交接模式允许将任务路由到合适的专业代理。

主要特点：

- 前台代理接收用户请求并进行分类
- 根据任务需求将对话交接给专门的代理
- 接管代理可以看到之前的对话历史

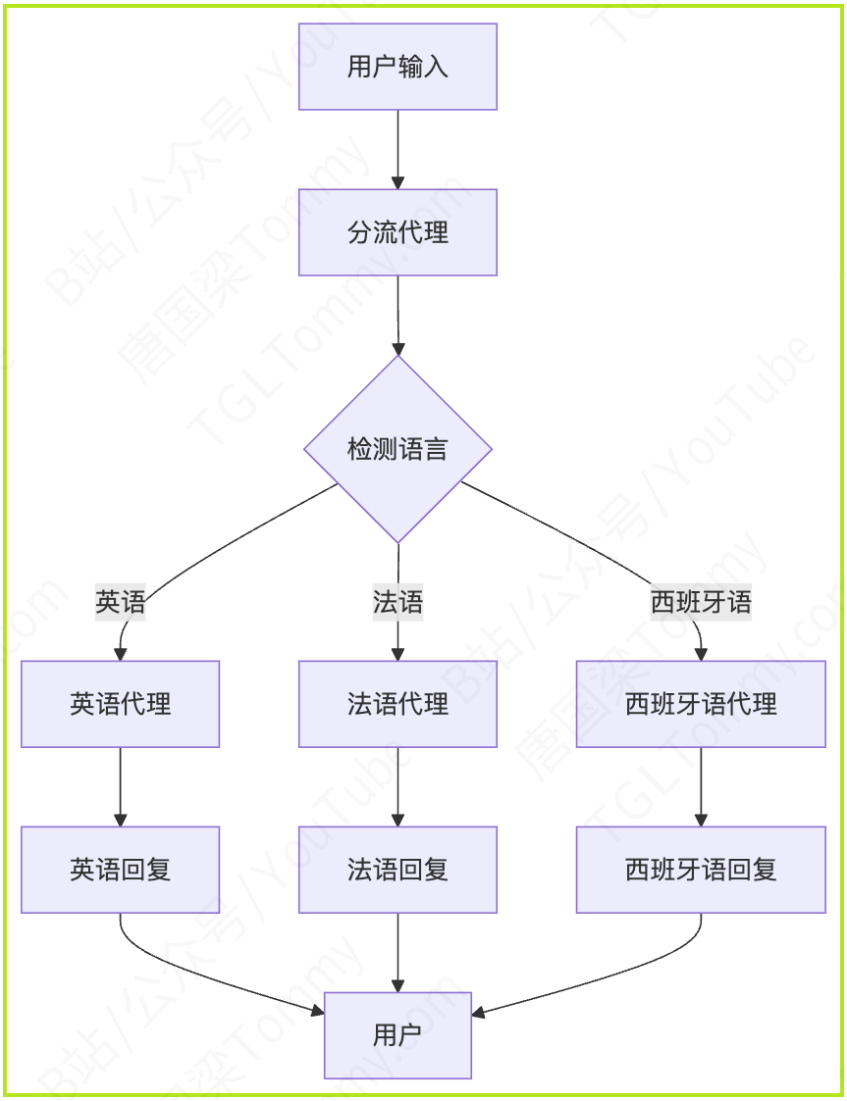
- 适合需要专业知识处理的场景

示例实现： `routing.py` 展示了一个语言路由系统：

- 前台代理分析用户使用的语言
- 将对话交接给相应的语言专家代理（法语、西班牙语或英语）
- 专家代理接管对话并用相应语言回复

适用场景：客户支持系统、多语言服务、需要不同专业知识领域的应用等。

流程图：



我需要优化附件中的README.md文件中关于"代理作为工具"模式的描述，使其与最新的脚本代码一致。

现在的脚本代码已经包含了以下翻译代理：

1. 西班牙语翻译代理 (spanish_agent)
2. 法语翻译代理 (french_agent)
3. 意大利语翻译代理 (italian_agent)
4. 中文翻译代理 (chinese_agent)

5. 英语翻译代理 (english_agent)

而现有的README.md描述只提到了西班牙语、法语和意大利语翻译代理。

另外，我们还使用了Ollama本地模型服务替代了默认的OpenAI API，这也是一个值得在描述中提及的更新。

因此，我需要优化这段描述，使其包含所有的翻译代理，并提及使用Ollama本地模型服务的情况。根据最新的脚本代码，我为您优化了描述：

3. 代理作为工具

与交接模式不同，代理作为工具模式保持原代理对对话的控制，仅调用其他代理作为工具来执行特定任务。

主要特点：

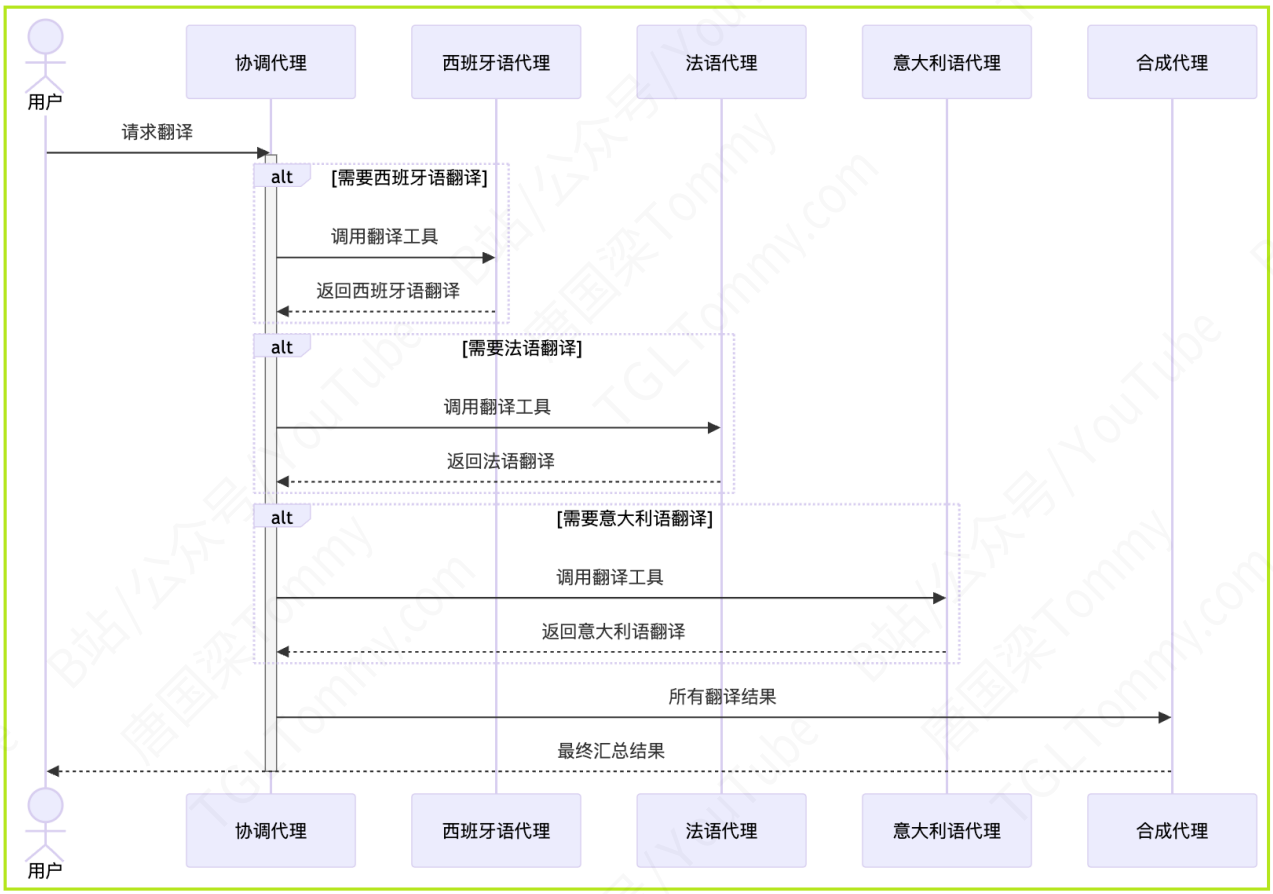
- 主代理保持对话控制权
- 工具代理独立运行并返回结果
- 可以并行调用多个工具代理
- 提供更灵活的工作流程设计
- 支持本地模型服务替代云API

示例实现： `agents_as_tool_ollama.py` 展示了一个多语言翻译系统：

- 主协调代理接收用户请求
- 根据需要调用不同的翻译工具（西班牙语、法语、意大利语、中文或英语翻译代理）
- 合成代理汇总所有翻译结果
- 使用本地Ollama模型服务替代OpenAI API，提供更灵活的部署选择

适用场景：需要组合多种功能的复杂任务，如多语言翻译、多模态分析等。也适用于需要本地部署、保护隐私或降低API成本的应用场景。

序列图：



4. LLM 作为评判者

LLM 常常可以通过反馈提高输出质量。这种模式使用一个模型生成内容，另一个模型评估内容并提供反馈。

主要特点：

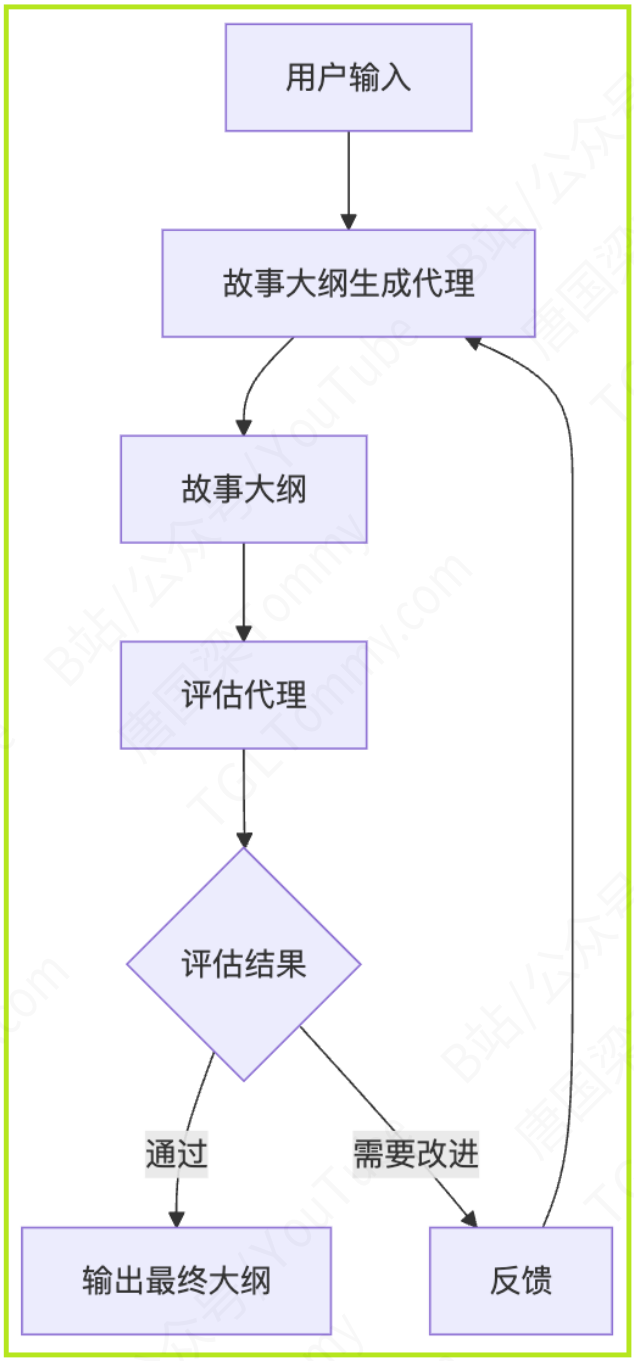
- 循环改进内容质量
- 使用结构化输出确保评估一致性
- 允许逐步优化生成的内容
- 可使用较小模型生成，大模型评估，优化成本

示例实现： `llm_as_a_judge.py` 展示了一个故事大纲改进流程：

1. 第一个代理生成故事大纲
2. 评估代理检查大纲并提供反馈
3. 如果评估未通过，将反馈发回第一个代理
4. 循环直到评估代理满意为止

适用场景：需要高质量输出的任务，如内容创作、代码生成、报告撰写等。

流程图：



5. 并行化

并行运行多个代理是一种常见模式，可以优化延迟或生成多个候选结果。

主要特点：

- 使用 `asyncio.gather` 并行执行多个代理
- 可以同时尝试多种方法解决问题
- 通过评选机制从多个结果中选出最佳答案
- 提高系统的响应速度和输出质量

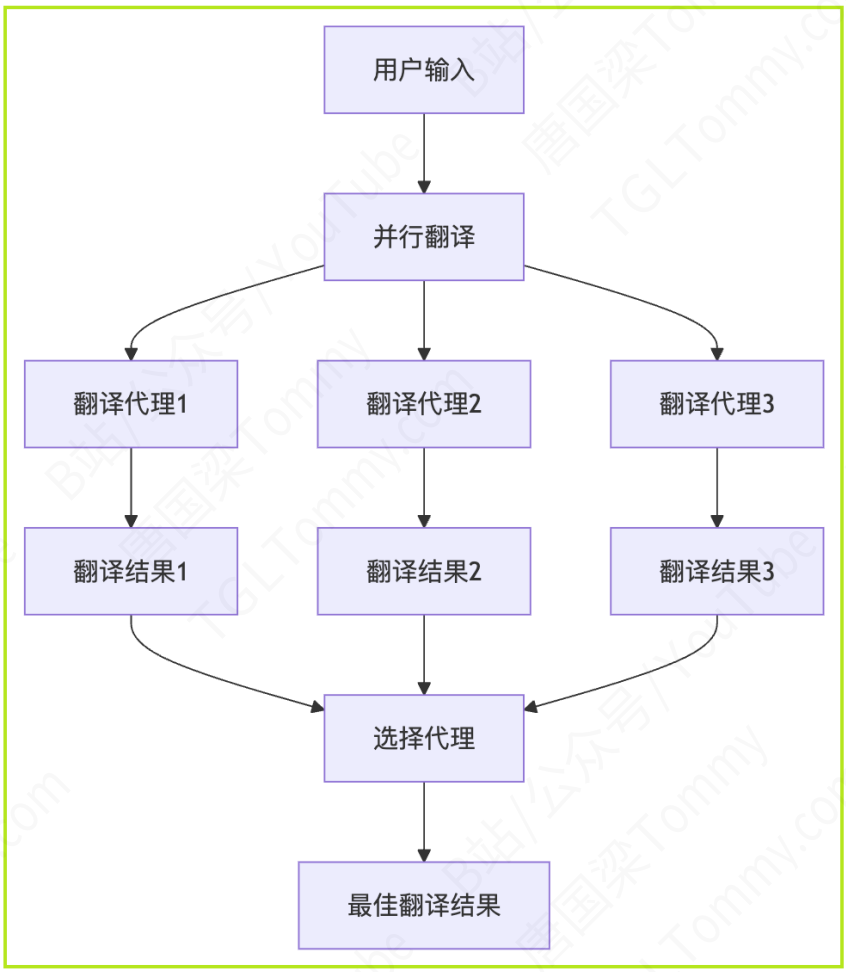
示例实现： `parallelization.py` 展示了并行翻译系统：

- 同时运行多个相同的翻译代理

- 每个代理独立生成翻译结果
- 最后使用选择代理从多个翻译中挑选最佳结果

适用场景：需要多种尝试的创意任务、对延迟敏感的应用、需要比较多个方案的决策系统等。

并行流程图：



6. 防护栏

防护栏是确保代理安全和合规的重要机制，可以应用于输入和输出。

输入防护栏

输入防护栏检查用户输入，确保其符合处理条件。

主要特点：

- 在代理处理前验证输入内容
- 使用"绊线"机制在问题出现时立即中断
- 可以快速拒绝不适合的请求
- 支持使用较快的模型进行初步筛查

示例实现： `input_guardrails.py` 展示了一个数学作业检测防护栏：

- 检查用户是否在要求代理完成数学作业
- 如果触发防护栏，立即拒绝请求并返回适当的消息

输出防护栏

输出防护栏检查代理生成的内容，确保其符合安全和政策要求。

主要特点：

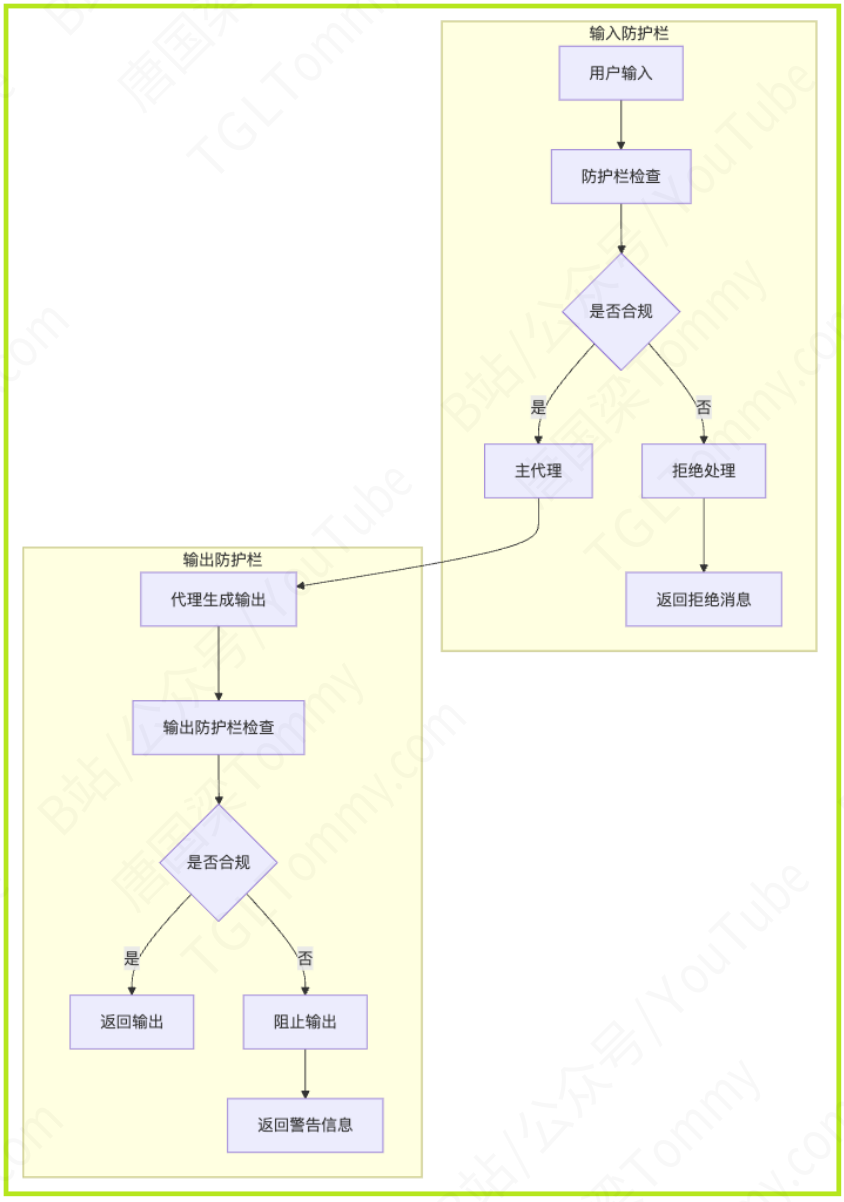
- 验证代理生成的输出内容
- 防止敏感信息泄露
- 确保回复符合内容政策

示例实现： `output_guardrails.py` 展示了一个敏感数据检测防护栏：

- 检查输出是否包含电话号码等敏感信息
- 如果发现敏感信息，触发防护栏并阻止输出

适用场景：需要内容审核的系统、处理敏感数据的应用、公共服务和教育平台等。

防护栏流程图：



组合使用多种模式

实际应用中，这些模式通常会组合使用，创建更复杂、更强大的代理系统。例如：

- 使用确定性流程分解任务，同时在每个步骤使用防护栏确保安全
- 结合路由和工具模式，创建既有专业知识又能调用外部工具的系统
- 使用并行化和LLM评判者，生成多个方案并选择最佳结果

通过灵活组合这些基本模式，可以构建适应各种需求的复杂AI代理系统。

组合模式示例：