OpenAI Agents SDK 项目概述

```
# 项目名称：openai-agents-python【v 0.0.5版本】
# git链接：https://github.com/openai/openai-agents-python.git
```

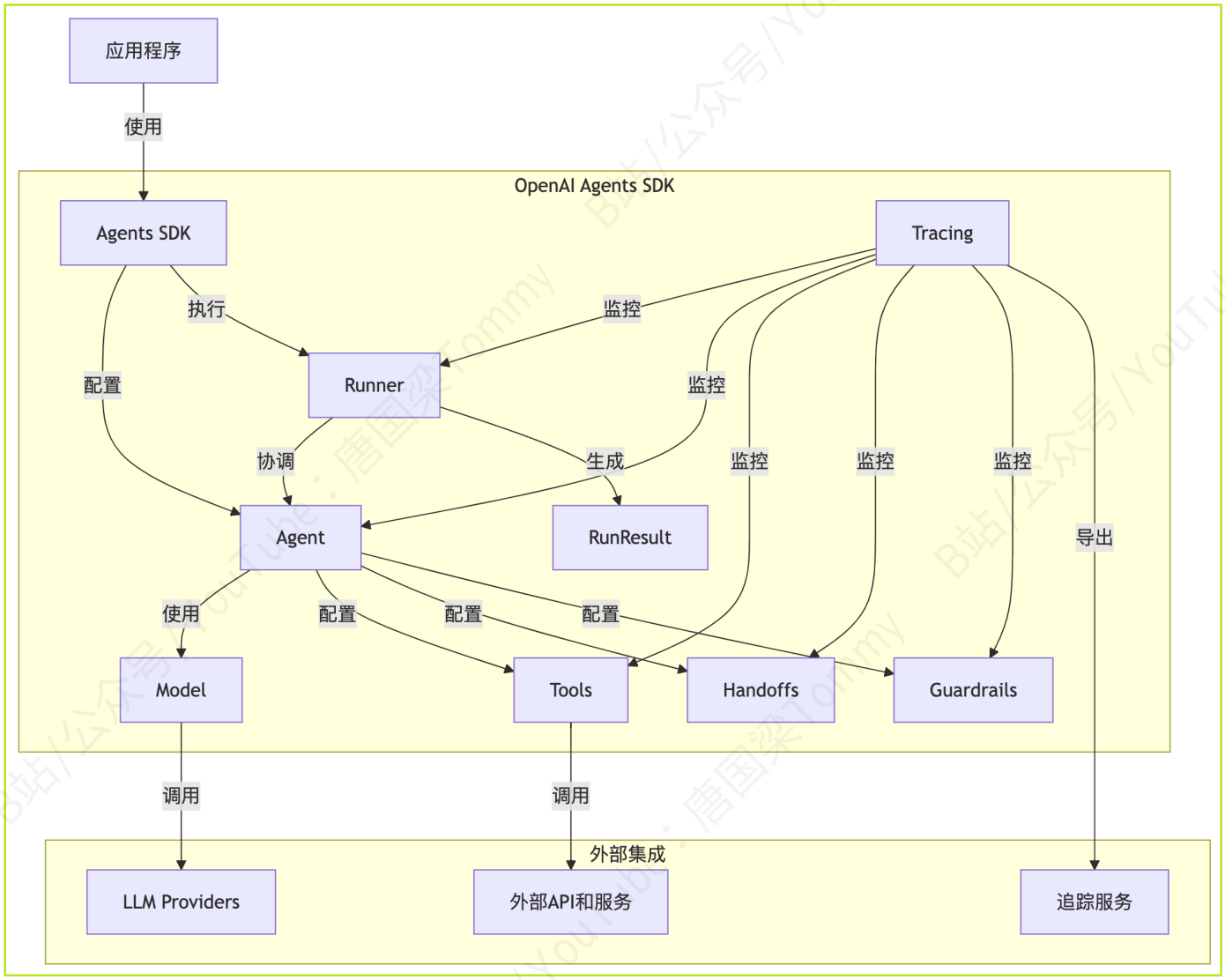
1. 项目概述

OpenAI Agents SDK 是一个轻量级但功能强大的框架，用于构建多智能体工作流(workflow)。它为开发者提供了一套工具，可以创建、配置和运行基于LLM的智能体系统，支持多智能体协作、工具调用、安全防护以及流程追踪等核心功能。

该框架采用分层架构设计：

- **模型层**：抽象不同LLM服务商的接口
- **代理层**：实现代理核心逻辑
- **工具层**：提供工具注册和执行机制
- **runtime 层**：管理代理执行流程
- **安全层**：实现护栏和安全控制
- **监控层**：提供跟踪和诊断功能

2. 核心模块



2.1 智能体（Agent）

智能体是该框架的核心概念，通过`Agent`类实现。每个智能体包含以下几个关键组件：

- **指令（Instructions）**：类似于"系统提示"，定义智能体的行为和响应方式
- **名称（Name）**：智能体的标识符
- **模型（Model）**：底层使用的语言模型，默认为OpenAI模型，但支持其他兼容的模型服务商
- **工具（Tools）**：智能体可以使用的功能工具集合
- **防护栏（Guardrails）**：用于验证输入和输出的安全检查
- **交接点（Handoffs）**：允许智能体将控制权转移给其他智能体

智能体实现了通用类型系统，支持传递上下文对象，使开发者可以在工具函数、交接、防护栏等组件之间共享状态。

2.2 运行器（Runner）

`Runner`类负责协调和执行智能体工作流。主要功能包括：

- 管理智能体循环（执行模型调用、处理工具调用、处理交接等）
- 处理并行执行的防护栏验证
- 提供同步和异步运行接口
- 支持流式输出

2.3 工具（Tools）

工具允许智能体与外部系统交互，实现更强大的功能。框架支持以下主要工具：

- **函数工具（FunctionTool）**：包装Python函数供智能体调用
- **文件搜索工具（FileSearchTool）**：允许LLM搜索向量存储
- **网络搜索工具（WebSearchTool）**：允许LLM搜索网络
- **计算机工具（ComputerTool）**：允许LLM控制计算机

创建工具的主要方式是使用`@function_tool`装饰器，它可以自动生成工具描述和参数模式。

2.4 交接（Handoffs）

交接机制允许智能体将控制权转交给其他更专业的智能体，实现了模块化设计。交接主要包含：

- 工具名称和描述（用于LLM识别何时应该使用交接）
- 输入JSON模式（定义交接所需的参数）
- 交接调用函数（根据输入参数返回目标智能体）
- 输入过滤器（可选，用于修改传递给下一个智能体的对话历史）

2.5 防护栏（Guardrails）

防护栏是一种安全机制，分为两类：

- **输入防护栏（InputGuardrail）**：在智能体执行之前验证输入
- **输出防护栏（OutputGuardrail）**：验证智能体的最终输出

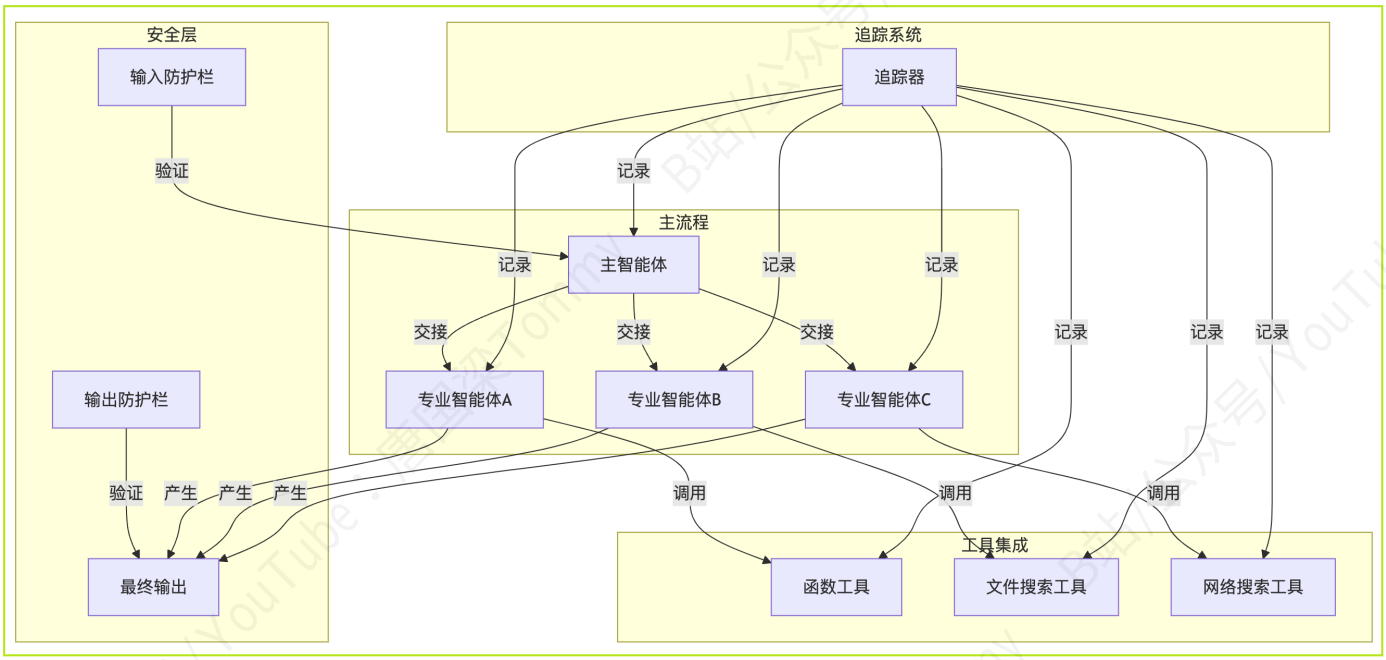
防护栏可以检查内容是否合规，如果触发警报，可以终止智能体的执行并引发异常。

2.6 追踪系统（Tracing）

追踪系统自动记录智能体运行的各个方面，便于调试和分析：

- 生成详细的跟踪信息，包括智能体调用、工具调用、交接等事件
- 支持自定义跨度（spans）和处理器（processors）

- 可以与多种外部跟踪系统集成（如Logfire, AgentOps, Braintrust等）



3. 主要特点

3.1 灵活的模型支持

框架支持任何符合OpenAI Chat Completions API格式的模型提供商。这包括：

- OpenAI模型（默认）
- 兼容的开源模型（ollama, transformers, sglang, vllm等）

3.2 强大的工具系统

工具系统使智能体能够执行各种操作：

- 自动从Python函数生成JSON模式
- 支持同步和异步函数
- 内置错误处理和验证
- 支持结构化输出

3.3 多智能体模式

框架支持多种智能体模式，包括：

- 确定性流程（如线性工作流）

- 迭代循环（智能体反复执行任务直到满足条件）
- 分支模式（基于条件选择不同路径）
- 嵌套智能体（智能体调用其他智能体作为工具）

3.4 上下文和状态管理

框架提供了上下文对象，用于在智能体运行过程中管理状态：

- 通过泛型类型系统确保类型安全
- 支持在工具、交接和防护栏之间共享状态
- 可以自定义上下文对象以适应特定应用需求

3.5 安全与控制

除了防护栏外，框架还提供其他安全控制功能：

- 最大回合限制，防止无限循环
- 模型行为错误检测
- 异常处理机制

4. 应用场景

根据项目结构和示例，该框架适用于多种场景：

4.1 客户服务

使用分流智能体根据请求类型将用户引导至专业智能体（客服、技术支持等）。

4.2 研究助手

构建能够搜索网络、进行信息整合和分析的研究助手。

4.3 工具使用

创建能够调用各种API和服务的智能体，如天气查询、数据处理等。

4.4 多轮对话系统

实现复杂的多轮对话，能够理解上下文并维持长期对话。

5. 项目代码

项目结构清晰，主要分为以下几个部分：

5.1 核心模块

agent.py

`Agent`类是整个框架的核心，代表一个能够执行指令并与用户交互的AI助手。

主要特点：

- 是一个泛型类，可关联自定义上下文对象（`TContext`）
- 核心属性包括：
  - `name`：代理名称
  - `instructions`：指导代理行为的系统提示
  - `handoff_description`：当作为交接目标时的描述
  - `model`和`model_settings`：配置LLM行为
- 可配置的功能组件：
  - `tools`：代理可使用的工具列表
  - `handoffs`：可交接到的其他代理
  - `input_guardrails`和`output_guardrails`：输入输出保护机制
  - `output_type`：指定输出的类型
- 提供`clone()`方法创建修改版本
- 提供`as_tool()`方法，将代理转换为可被其他代理调用的工具

run.py

`Runner`类负责执行代理逻辑，是整个执行流程的控制中心。

主要功能：

- 初始化和管理代理执行的上下文环境
- 处理输入处理和输出生成
- 执行代理调用的工具（同步/异步）
- 管理代理之间的交接流程
- 实现输入/输出guardrail的检查
- 支持流式输出结果
- 处理模型行为异常和错误

- 跟踪和记录执行过程

_run_impl.py

实现了一个高度模块化的代理（Agent）执行引擎，采用了一种"编排"思想，将模型响应处理、工具执行和代理交互等复杂流程拆分为清晰的步骤。

- 实现代理执行的底层核心逻辑
- 包含工具运行、处理模型响应和执行handoff的具体实现
- 管理执行流程中的事件处理和状态转换
- 处理追踪上下文和错误捕获

tool.py

定义了代理可以使用的各种工具类型和接口。

主要组件：

- `Tool` 抽象基类，定义工具的基本接口
- 具体工具实现：
  - `FunctionTool`：封装普通函数为工具
  - `ComputerTool`：与计算机交互的工具
  - `FileSearchTool`：文件搜索工具
  - `WebSearchTool`：网络搜索工具
- `function_tool` 装饰器：快速将函数转换为工具
- 工具错误处理机制：定义了统一的错误处理方式

handoffs.py

实现代理之间的任务转交机制，使一个代理可以将对话转交给更专业的代理。

核心概念：

- `Handoff` 类：表示从一个代理到另一个代理的转交操作
- `HandoffInputData`：封装传递给下一个代理的数据
- `HandoffInputFilter`：过滤传递给下一个代理的输入
- `handoff` 工厂函数：简化创建交接的过程
- 支持带参数和不带参数的交接方式
- 支持在交接时执行自定义逻辑

computer.py

定义了与计算机系统交互的抽象接口。

主要内容：

- 两个抽象基类：
  - `Computer`：同步操作接口
  - `AsyncComputer`：异步操作接口
- 支持的环境类型：`Environment = Literal["mac", "windows", "ubuntu", "browser"]`
- 支持的按钮类型：`Button = Literal["left", "right", "wheel", "back", "forward"]`
- 核心操作方法：
  - `screenshot`：获取屏幕截图
  - `click/double_click`：鼠标点击操作
  - `type`：键盘输入文本
  - `keypress`：按键操作
  - `scroll/move/drag`：鼠标移动和拖拽操作

guardrail.py

实现安全保护机制，确保代理的输入和输出符合预期规范。

主要组件：

- `InputGuardrail`：检查和验证输入数据
- `OutputGuardrail`：检查和验证输出数据
- `GuardrailFunctionOutput`：保护机制的执行结果
- 两个关键装饰器：
  - `input_guardrail`：创建输入保护机制
  - `output_guardrail`：创建输出保护机制
- 触发机制：当检测到违规情况时可以触发`tripwire`，中断代理执行
- 支持同步和异步保护函数

5.2 辅助模块

agent_output.py

- 定义`AgentOutputSchema`类，管理代理输出的JSON模式和验证
- 支持纯文本和结构化JSON输出

- 提供输出类型验证和错误处理
- 确保模型生成的输出符合预期格式

result.py

- 定义`RunResult`和`RunResultStreaming`类，封装代理执行结果
- 包含原始输入、生成的项目和最终输出
- 支持从结果中提取文本和其他信息
- 处理流式输出的状态管理

function_schema.py

- 定义`FuncSchema`类，封装函数的描述、参数和返回值
- 提供从Python函数自动生成JSON Schema的能力
- 支持从函数文档字符串提取描述信息
- 将验证后的数据转换为函数调用参数

items.py

- 定义运行过程中生成的各种项目类型
- 包括消息、工具调用、工具输出、推理过程等
- 提供项目转换和文本提取的辅助方法
- 管理模型响应的结构化表示

lifecycle.py

- 定义`RunHooks`和`AgentHooks`接口
- 提供代理生命周期事件的回调机制
- 支持监听代理启动、结束、工具使用和交接过程
- 允许自定义代理行为和流程

model_settings.py

- 定义`ModelSettings`类，配置LLM调用参数
- 管理温度、top_p、惩罚和截断等参数
- 支持参数覆盖和合并
- 适配不同模型提供商的设置需求

stream_events.py

- 定义流式输出时的事件类型
- 支持代理更新、原始响应和运行项目流
- 提供统一的事件接口

run_context.py

- 定义`RunContextWrapper`类，封装执行上下文
- 支持在代理、工具和guardrail之间共享状态
- 提供类型安全的上下文访问

exceptions.py

- 定义框架中使用的异常类型
- 包括`AgentsException`基类和各种特定异常
- 处理最大轮次超出、模型行为错误和用户错误等情况
- 支持guardrail触发的异常处理

_config.py

- 管理全局配置设置
- 处理OpenAI API密钥和客户端配置
- 控制默认API选择(chat completions或responses)

_debug.py

- 提供调试辅助功能
- 控制模型数据和工具数据的日志记录
- 通过环境变量启用或禁用调试功能

usage.py

- 定义`Usage`类，跟踪代理使用的Token数量
- 记录提示和完成的Token计数
- 支持使用量合并

strict_schema.py

- 增强JSON Schema的严格验证
- 确保模型生成的JSON符合预期结构
- 支持schema的转换和增强

---

models/目录

- 包含不同模型提供商的实现
- `interface.py` 定义模型接口和追踪配置
- `openai_chatcompletions.py` 实现基于聊天完成API的模型
- `openai_responses.py` 实现基于响应API的模型
- `openai_provider.py` 提供OpenAI模型实例化和管理

tracing/目录

- 实现执行过程的跟踪功能
- 定义 `Span` 和 `Trace` 表示执行跟踪单元
- 支持记录代理、工具和guardrail的执行情况
- 提供跟踪数据的导出和处理

extensions/目录

- 提供额外的功能扩展
- `handoff_filters.py` 包含常用的handoff输入过滤器
- `handoff_prompt.py` 提供推荐的handoff相关提示模板

这些模块共同构成了一个完整的代理系统架构，提供从模型调用、工具执行到结果处理的全流程支持，使开发者能够构建复杂而可靠的AI代理应用。

---

6. 总结

OpenAI Agents SDK 提供了一个全面而灵活的框架，用于构建基于LLM的智能体系统。它的核心优势在于：

1. **模块化设计**：通过智能体、工具、交接等组件实现关注点分离
2. **灵活性**：支持多种模型、工具和模式
3. **安全性**：内置防护栏和安全机制

4. **可追踪性**：详细的追踪系统便于调试和分析
5. **易用性**：直观的API和丰富的文档

这个框架非常适合构建复杂的智能体系统，从简单的对话助手到复杂的多智能体协作工作流，为开发者提供了强大而灵活的工具集。