# LangSmith

LangSmith 是一个用于构建生产级 LLM 应用程序的平台。监控和评估您的应用程序，以便您可以快速、自信地交付。

**LangSmith 与框架无关——你可以在有或没有 LangChain 的开源框架 langchain 和 langgraph 的情况下使用它。**

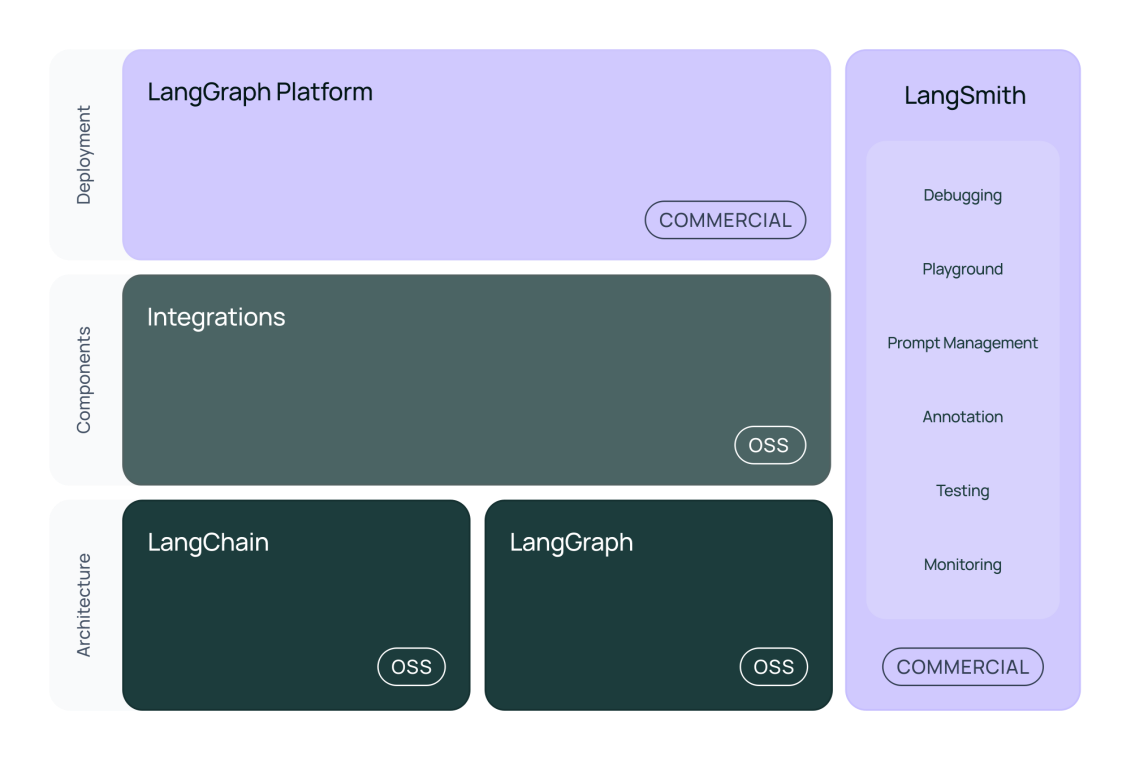
可观测性是使用大型语言模型（LLM）构建的应用程序的关键要求。LLM是不确定的，这意味着相同的提示可以产生不同的响应。这种行为使调试和监控比传统软件更具挑战性。

LangSmith通过提供对应用程序如何处理请求的端到端可见性来解决这一问题。每个请求都会生成一个追踪记录，其中完整记录了整个处理过程。在单个追踪记录中包含多个独立运行单元，即应用程序执行的具体操作（如LLM调用或检索步骤）。通过追踪这些运行单元，您可以检查、调试和验证应用程序的行为表现。

支持可视化的追踪、调试和监控LangChain应用，应用表现不佳时可以清晰看到每步的输入和输出，以及模型究竟在想什么，可用于优化提示词、分析错误、提升应用的性能。

# LangChain

Python包langchain和公司LangChain的目标是尽可能让开发者轻松构建具备推理能力的应用程序。虽然LangChain最初只是一个单一的开源包，但它已发展成为一家公司和一个完整的生态系统。



LangChain：

教程：https://python.langchain.com/docs/how\_to/

<https://python.langchain.ac.cn/docs/how_to/>

LLM和提示词（Prompt）

链（Chain）

LCEL表达式

检索增强生产（RAG）

Agent智能体

Memory模型记忆

# LangGraph

LangGraph 是 LangChain 的一个扩展，旨在通过将步骤建模为图中的边和节点，构建健壮且有状态的基于LLM的**多智能体应用程序**。

（LangGraph is an extension of LangChain aimed at building robust and stateful multi-actor applications with LLMs by modeling steps as edges and nodes in a graph.）

**基本概念：**

LangGraph 的核心是将代理工作流建模为图。可以使用三个关键组件来定义代理的行为：

State：一个共享数据结构，表示应用程序的当前快照。它可以是任何 Python 类型，但通常是 TypedDict 或 Pydantic BaseModel。

Nodes：编码代理逻辑的 Python 函数。它们接收当前的 State 作为输入，执行一些计算或副作用，并返回一个更新的 State。

Edges：Python 函数，根据当前的 State 决定接下来执行哪个 Node。它们可以是条件分支或固定转换。

通过组合 Nodes 和 Edges，可以创建复杂、循环的工作流，使 State 随时间演进。然而，真正的力量来自于 LangGraph 如何管理 State。需要强调的是：Nodes 和 Edges 不过是 Python 函数——它们可以包含一个 LLM，也可以只是普通的 Python 代码。

LangGraph 的底层图算法使用消息传递来定义通用程序。当一个节点完成其操作时，它会沿着一条或多条边向其他节点发送消息。这些接收节点随后执行其函数，将结果消息传递给下一组节点，然后该过程继续。受 Google 的 Pregel 系统启发，程序以离散的“超步”进行。

超步可以被认为是图节点的一次迭代。并行运行的节点属于同一个超步，而顺序运行的节点属于不同的超步。在图执行开始时，所有节点都处于 inactive 状态。当节点在其任何传入边（或“通道”）上接收到新消息（状态）时，它就会变为 active。活跃节点随后运行其函数并返回更新。在每个超步结束时，没有传入消息的节点通过将自身标记为 inactive 来投票 halt（停止）。当所有节点都处于 inactive 状态且没有消息传输时，图执行终止。

StateGraph

StateGraph 类是主要的图类。它通过用户定义的 State 对象进行参数化。

要构建图，您首先定义状态，然后添加节点和边，最后编译它。究竟什么是编译图，为什么需要它？编译是一个相当简单的步骤。它对图的结构进行一些基本检查（例如，没有孤立节点）。它也是您可以指定运行时参数的地方，例如检查点和断点。您只需调用 .compile 方法即可编译图。

graph = graph\_builder.compile(...)

您必须先编译图才能使用它。

**核心优势：**

1. 持久执行：构建能够经受住故障并能长时间运行的智能体，能从中断处自动恢复。

LangGraph 有一个内置的持久化层，通过检查点器实现。当你使用检查点器编译图时，检查点器会在每个超级步骤保存图状态的 checkpoint。这些检查点保存到 thread 中，可以在图执行后访问。因为 threads 允许在执行后访问图的状态，所以人机协作、记忆、时间旅行和容错等多种强大功能都成为可能。

**检查点：**

检查点是每个超级步骤保存的图状态的快照，由具有以下关键属性的 StateSnapshot 对象表示

config：与此检查点关联的配置。

metadata：与此检查点关联的元数据。

values：此时状态通道的值。

next：要在图中接下来执行的节点名称的元组。

tasks：包含要执行的后续任务信息的 PregelTask 对象元组。如果该步骤之前曾尝试过，它将包含错误信息。如果图在节点内部被动态中断，tasks 将包含与中断相关的附加数据。

示例如下：



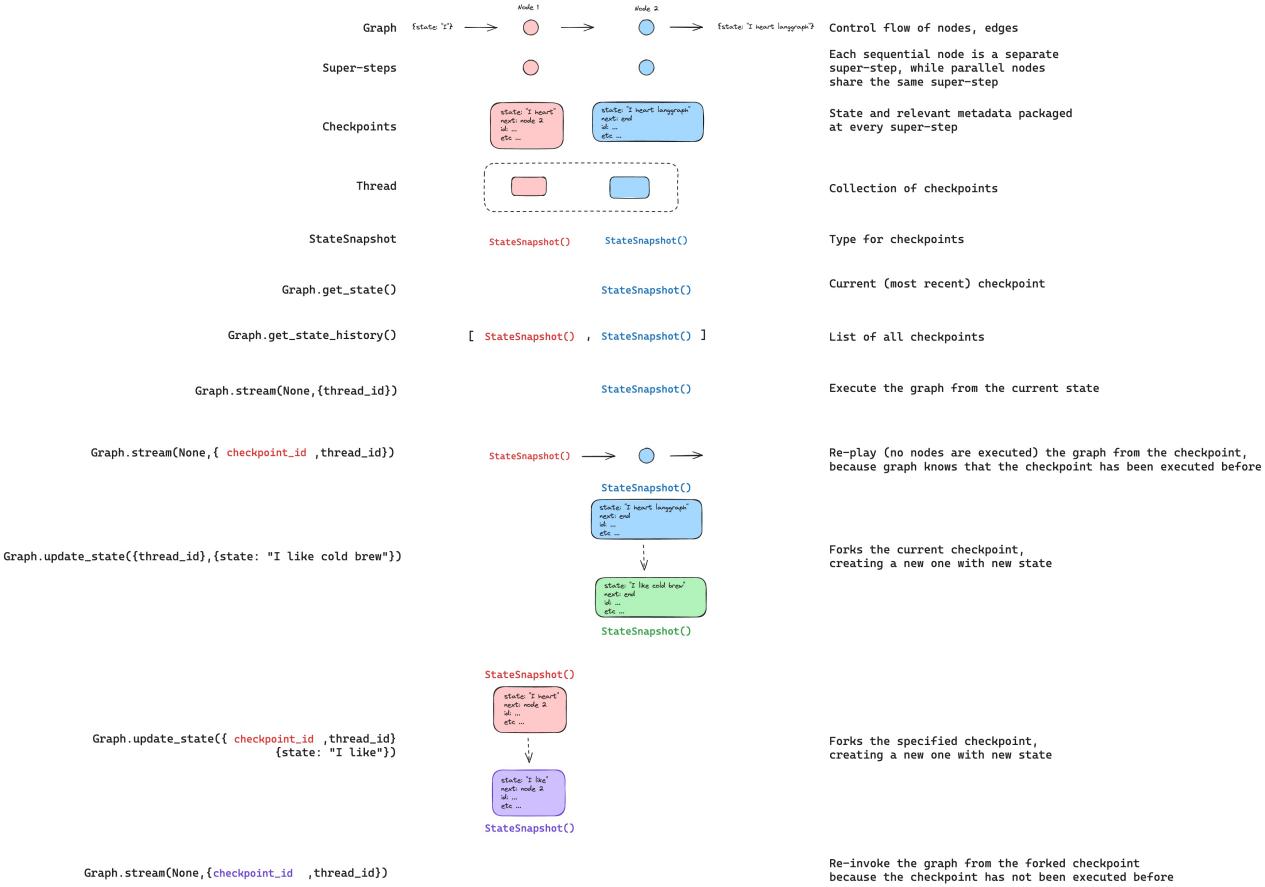
作用：

1. 获取状态历史config = {"configurable": {"thread\_id": "1"}} list(graph.get\_state\_history(config))
2. 回放只执行检查点之后的步骤

config = {"configurable": {"thread\_id": "1", "checkpoint\_id": "0c62ca34-ac19-445d-bbb0-5b4984975b2a"}}

graph.invoke(None, config=config)

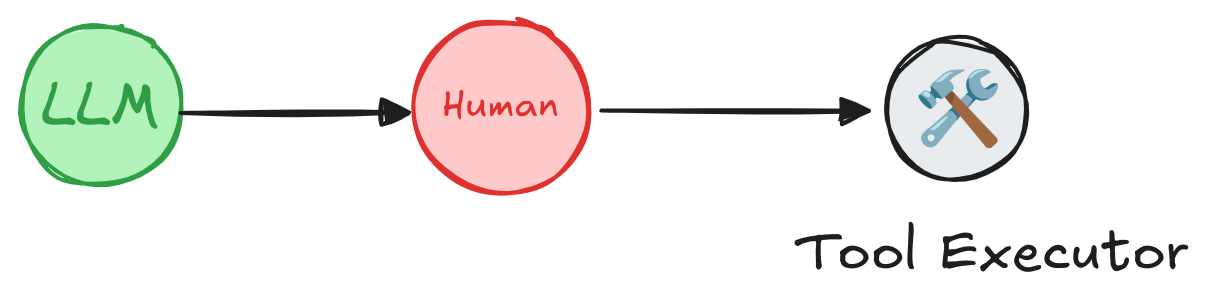
1. 更新状态：除了从特定的checkpoints回放图之外，还可以编辑图的状态，使用graph.update\_state()实现。接收两个参数分别是config，应包含thread\_id和checkpoint\_id字段。values应包含要更新的状态值



（2）人机协同：在执行过程中的任何时刻，通过检查和修改智能体状态，无缝地引入人类监督。

资料：

https://github.langchain.ac.cn/langgraph/how-tos/human\_in\_the\_loop/add-human-in-the-loop/#review-tool-calls



支持在工作流中审查、编辑、批准工具的调用，以便在工作流的任何时刻进行人工干预。这在大型语言模型 (LLM) 驱动的应用中尤其有用，因为模型输出可能需要验证、修正或额外的上下文。  
 要审查、编辑和批准智能体或工作流中的工具调用，请使用中断来暂停图并等待人工输入。中断使用 LangGraph 的持久化层，它会保存图的状态，从而无限期地暂停图的执行，直到您恢复为止。

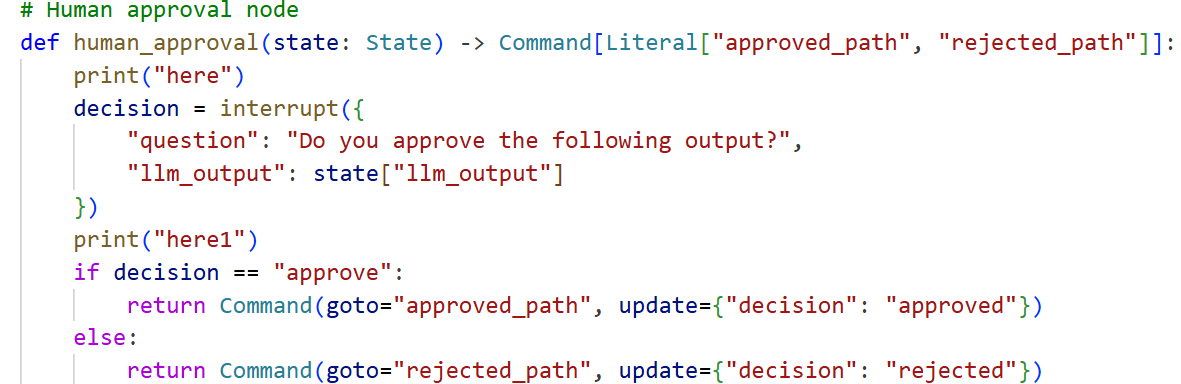
动态中断Interrupt

最终要的机制是Interrupt中断和Command恢复机制。

当在图中使用 interrupt 函数时，执行会在此处暂停并等待用户输入。

要恢复执行，请使用 Command 原语，它可以通过 invoke 或 stream 方法提供。图会从最初调用interrupt(...)的节点的开头恢复执行。这一次，interrupt函数将返回在Command(resume=value)中提供的值，而不会再次暂停。从节点开头到interrupt的所有代码都将被重新执行。

例子：





静态中断：Interrupt\_before Interrupt\_after,静态中断不推荐人在环路工作流，静态中断推荐用于调试，在此先不介绍。

（3）全面记忆：创建真正有状态的智能体，既有用于持续推理的短期工作记忆，也有跨会话的长期持久记忆。

短期记忆，或线程范围的记忆，可以随时从单个用户对话线程中召回。LangGraph将短期记忆作为代理状态的一部分进行管理。状态通过检查点持久化到数据库中，以便线程可以随时恢复。短期记忆在图被调用或步骤完成时更新，并在每个步骤开始时读取状态。

长期记忆在对话线程之间共享。它可以在任何时候和任何线程中召回。记忆可以限定在任何自定义命名空间，而不仅仅是单个线程ID。

修剪消息、总结消息、删除消息等

（4）使用 LangSmith 调试：通过可视化工具深入了解复杂的智能体行为，这些工具可以追踪执行路径、捕获状态转换并提供详细的运行时指标。

（5）生产级部署：使用专为处理有状态、长期运行工作流的独特挑战而设计的可扩展基础设施，自信地部署复杂的智能体系统。