LangChain 模块解析（下）

LangChain Chains Hands-on

Press Space for next page →

《AGI 应用实践》系列教程

AGI OVERVIEW

- 101：AGI 提示工程指北
- 102：OpenAI API 浅出

LANGCHAIN IN ACTION

- 201：LangChain 功能模块解析（上篇）
- 202：LangChain 功能模块解析（中篇）
- 203：LangChain 功能模块解析（下篇）

教程大纲

- Agent vs. Chain
- Agent 的思考链模式：ReAct
- Agent 的思考链模式：Plan and Execute
- ReAct 模式的工具能力强化：JSON Schema
- ReAct 模式的工具能力强化：OpenAI Functions
- ReAct 模式下的 XML Agent
- Agent 的工具箱
- Agent 工具箱中的工具集
- 必不可缺的 Callback 回调系统

Agent vs. Chain

如果说 Agent 是 Chain 的进化形态，那哪些方面是对等的？哪些地方又存在质变？

➡️ Agent 继承了 Chain 的所有能力

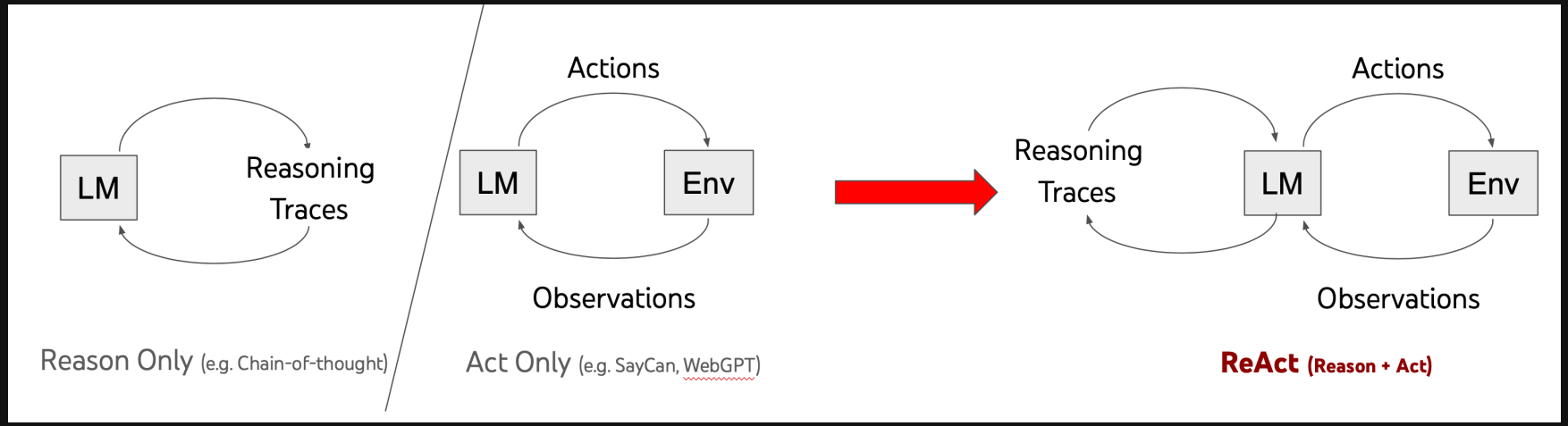
- Chain 的能力构成了 Agent 的“推理链”能力基础
  - Agent 的执行器（Executor）就是基于 LLM Chain 实现的
- Agent 自身具备和 Chain 类似的“调用 / 执行”接口的能力

⬆️⬆️ Agent 在 Chain 的能力基础上进化出了两个核心能力

- 一条思考链（[W]Chain of Thought）：Agent 可以规划和调度推理链，会思考分步骤的、每一步的行动
- 一个工具箱（Toolbox）：Agent 自带工具体系，可以接入并调用外部工具（工具可以按既定接口规范定制）
  - Agent 只关注每个工具的输入输出（格式及内容），工具内部可以自行进行本地和 Web 接口调用

Agent 的思考链模式：ReAct

ReAct: Synergizing Reasoning and Acting in Language Models



ReAct 使用大语言模型同时生成推理链和行动，两者交替出现，相互支持：

- 推理链帮助 Agent 制定、跟踪和更新行动计划
- 行动让 Agent 与环境上下文交互，获取更多信息支持推理

♻️ 流程概览：给出任务 ➡️ 推理下一步行动 ➡️ 行动并更新环境 ➡️ 观察并推理下一步行动 ➡️ 循环直至任务完成

使用 ReAct 模式的 Agent

```
import { initializeAgentExecutorWithOptions } from "langchain/agents";
import { OpenAI } from "langchain/llms/openai";
import { SerpAPI } from "langchain/tools";
import { Calculator } from "langchain/tools/calculator";

const model = new OpenAI({ temperature: 0 });
const tools = [
  /* Web 搜索工具 */
  new SerpAPI(process.env.SERPAPI_API_KEY, {
    location: "Austin,Texas,United States",
    hl: "en",
    gl: "us",
  }),
  /* 本地计算器工具 */
  new Calculator(),
];

const executor = await initializeAgentExecutorWithOptions(tools, model, {
  agentType: "zero-shot-react-description",
  verbose: true,
});

const input = `Who is Olivia Wilde's boyfriend? What is his current age raised to the 0.23 power?`;
```

Run: AgentExecutor

Playground

Trace

- AgentExecutor • Success
  - Chain LLMChain
    - LLM OpenAI
  - Tool SerpAPI
  - Chain LLMChain
    - LLM OpenAI
  - Tool Calculator
  - Chain LLMChain
    - LLM OpenAI
  - Tool SerpAPI
  - Chain LLMChain
    - LLM OpenAI

AgentExecutor

Run | Feedback | Metadata

INPUT Copy

```
1  input: Who is Olivia Wilde's boyfriend? What is his current age raised to the 0.23 power?
```

YAML

OUTPUT Copy

Harry Styles' age raised to the 0.23 power is 2.169459462491557.

ReAct 模式的核心提示词

```
export const PREFIX = `Answer the following questions as best you can. You have access to the following tools:`;

export const FORMAT_INSTRUCTIONS = `Use the following format in your response:

Question: the input question you must answer
Thought: you should always think about what to do
Action: the action to take, should be one of [{tool_names}]
Action Input: the input to the action
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can repeat N times)
Thought: I now know the final answer
Final Answer: the final answer to the original input question`;

export const SUFFIX = `Begin!

Question: {input}
Thought:{agent_scratchpad}`;
```

```
const executor = await initializeAgentExecutorWithOptions(tools, chat, {
  agentType: "<certain-agent-type>",
  agentArgs: { prefix, suffix },  // 可自定义前置和后置提示词
});
```

Agent 的思考链模式：Plan and Execute

Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models

顾名思义，Agent 通过首先计划要做什么，然后执行子任务来实现目标

- 这个想法很大程度上是受到 BabyAGI 和后来的《Plan-and-Solve Prompting》论文的启发
- **计划几乎总是由 LLM 完成；执行通常由单独的（带有工具的）Agent 完成**

该 Agent 有两个主要的关键步骤：

1. 首先，Agent 使用 LLM 创建一个计划，以明确的步骤回答查询
2. 一旦制定了计划，它就会使用传统的 Agent（比如 ReAct）来解决每个步骤

> 其想法是，计划步骤通过将较大的任务分解为更简单的子任务，使 LLM 更加“步入正轨”。
> 但是，与传统 Agent 相比，此方法需要更多单独的 LLM 查询，并且延迟更高。

使用 Plan and Execute 模式的 Agent

该模式的 Agent 目前仅支持 Chat 模型

```
import { Calculator } from "langchain/tools/calculator";
import { SerpAPI } from "langchain/tools";
import { ChatOpenAI } from "langchain/chat_models/openai";
import { PlanAndExecuteAgentExecutor } from "langchain/experimental/plan_and_execute";

const tools = [new Calculator(), new SerpAPI()];
const model = new ChatOpenAI({
  temperature: 0,
  modelName: "gpt-3.5-turbo",
  verbose: true,
});

const executor = PlanAndExecuteAgentExecutor.fromLLMAndTools({
  llm: model,
  tools,
});

const result = await executor.call({
  input: `Who is the current president of the United States? What is their current age raised to the second power?`,
});
```

Run: PlanAndExecuteAgentExe...

Playground

Trace

- PlanAndExecuteAgentExecutor • Success
  - Chain LLMChain
  - Chain AgentExecutor
  - Chain AgentExecutor
  - Chain AgentExecutor
  - Chain AgentExecutor

PlanAndExecuteAgentExecutor

Run | Feedback | Metadata

INPUT Copy

```
1  input: Who is the current president of the
   United States? What is their current age
   raised to the second power?
```

YAML

OUTPUT Copy

The current president of the United States is Joe Biden. His current age raised to the second power is 6400.

Plan and Execute 模式的核心提示词

```
export const PLANNER_SYSTEM_PROMPT_MESSAGE_TEMPLATE = [
  `Let's first understand the problem and devise a plan to solve the problem.`,
  `Please output the plan starting with the header "Plan:"`,
  `and then followed by a numbered list of steps.`,
  `Please make the plan the minimum number of steps required`,
  `to answer the query or complete the task accurately and precisely.`,
  `Your steps should be general, and should not require a specific method to solve a step. If the task is a question,`,
  `the final step in the plan must be the following: "Given the above steps taken,`,
  `please respond to the original query."`,
  `At the end of your plan, say "<END_OF_PLAN>"`,
].join(" ");

export const DEFAULT_STEP_EXECUTOR_HUMAN_CHAT_MESSAGE_TEMPLATE = `Previous steps: {previous_steps}

Current objective: {current_step}

{agent_scratchpad}

You may extract and combine relevant data from your previous steps when responding to me.`;
```

ReAct 模式的工具能力强化：JSON Schema

工具的输入输出限定为单个字符串 ➡️ 可以使用工具提供的 JSON Schema 接口定义来扩展输入输出

```
import { z } from "zod";
import { ChatOpenAI } from "langchain/chat_models/openai";
import { initializeAgentExecutorWithOptions } from "langchain/agents";
import { Calculator } from "langchain/tools/calculator";
import { DynamicStructuredTool } from "langchain/tools";

const model = new ChatOpenAI({ temperature: 0 });
const tools = [
  new Calculator(), // Older existing single input tools will still work
  new DynamicStructuredTool({
    name: "random-number-generator",
    description: "generates a random number between two input numbers",
    schema: z.object({
      low: z.number().describe("The lower bound of the generated number"),
      high: z.number().describe("The upper bound of the generated number"),
    }),
    func: async ({ low, high }) =>
      (Math.random() * (high - low) + low).toString(), // Outputs still must be strings
    returnDirect: false, // This is an option that allows the tool to return the output directly
  }),
];
```

Run: AgentExecutor

Playground

Trace

- AgentExecutor • Success
  - Chain LLMChain
    - LLM ChatOpenAI
  - Tool DynamicStructuredTool
  - Chain LLMChain
    - LLM ChatOpenAI
  - Tool Calculator
  - Chain LLMChain
    - LLM ChatOpenAI

AgentExecutor

Run | Feedback | Metadata

INPUT Copy

```
1  input: What is a random number between 5 and
   10 raised to the second power?
```

YAML

OUTPUT Copy

The random number between 5 and 10 raised to the second power is 26.052322060418064.

ReAct 模式的工具能力强化：OpenAI Functions

OpenAI 模型（如 `gpt-3.5-turbo-0613`）已经过微调，可以检测何时应调用函数并响应应传递给函数的输入

- OpenAI Functions 的目标是比通用文本完成或聊天 API 更可靠地返回有效且有用的函数调用
- 在 API 调用中，您可以描述函数并让模型智能地选择输出包含调用这些函数的参数的 JSON 对象

```
import { initializeAgentExecutorWithOptions } from "langchain/agents";
import { ChatOpenAI } from "langchain/chat_models/openai";
import { SerpAPI } from "langchain/tools";
import { Calculator } from "langchain/tools/calculator";

const tools = [new Calculator(), new SerpAPI()];
const chat = new ChatOpenAI({ modelName: "gpt-4", temperature: 0 });

const executor = await initializeAgentExecutorWithOptions(tools, chat, {
  agentType: "openai-functions",
  verbose: true,
});

const result = await executor.run("What is the weather in New York?");
```

Run: AgentExecutor

Playground

Trace

- AgentExecutor • Success
  - LLM ChatOpenAI
  - Tool SerpAPI
  - LLM ChatOpenAI

AgentExecutor

Run | Feedback | Metadata

INPUT

Copy

```yaml
input: What is the weather in New York?
chat_history:
  - id:
      - langchain
      - schema
      - HumanMessage
    lc: 1
    type: constructor
    kwargs:
      content: What is the weather in New York?
      additional_kwargs: {}
```

YAML

ReAct 模式下的 XML Agent

某些语言模型（例如 Anthropic 的 Claude）特别擅长推理/编写 XML

```
import { ChatAnthropic } from "langchain/chat_models/anthropic";
import { initializeAgentExecutorWithOptions } from "langchain/agents";
import { SerpAPI } from "langchain/tools";

const model = new ChatAnthropic({ modelName: "claude-2", temperature: 0.1 });
const tools = [new SerpAPI()];

const executor = await initializeAgentExecutorWithOptions(tools, model, {
  agentType: "xml",
  verbose: true,
});
console.log("Loaded agent.");

const input = `What is the weather in Honolulu?`;
const result = await executor.call({ input });
```

Run: AgentExecutor

Playground

Trace

- AgentExecutor • Success
  - Chain LLMChain
    - LLM ChatAnthropic
  - Tool SerpAPI
  - Chain LLMChain
    - LLM ChatAnthropic

AgentExecutor

Run | Feedback | Metadata

INPUT Copy

```
1 input: What is the weather in Honolulu?
```

YAML

OUTPUT Copy

The weather in Honolulu is currently 75 degrees Fahrenheit with a small craft advisory in effect. The forecast calls for generally clear skies tonight with a low of 75 degrees.

Agent 的工具箱

工具是 Agent 可以用来与世界交互的功能

Agent 只关心工具的输入输出，内部实现对 Agent 透明

- 这些工具可以是通用实用程序（例如搜索）、其它 Chain 调用，甚至其它 Agent 调用

Agent 定义了工具的标准接口，以实现无缝集成

- 具体来说，工具的接口具有单个文本输入和单个文本输出
- 它包括名称和描述，用于向模型传达该工具的用途以及何时使用它

```
abstract class Tool {
  abstract name: string;
  abstract description: string;

  abstract _call(arg: string): Promise<string>;
}
```

Calculator 工具的实现

```ts
import { Parser } from "expr-eval";
import { Tool } from "./base.js";

/**
 * The Calculator class is a tool used to evaluate mathematical
 * expressions. It extends the base Tool class.
 */
export class Calculator extends Tool {
  name = "calculator";

  description = `Useful for getting the result of a math expression.
  The input to this tool should be a valid mathematical expression that could be executed by a simple calculator.`;

  /** @ignore */
  async _call(input: string) {
    try {
      return Parser.evaluate(input).toString();
    } catch (error) {
      return "I don't know how to do that.";
    }
  }
}
```

Tool 的结构化输入定义

Tool 可以通过 JSON Schema 来强化输入的定义，在 JS/TS SDK 中会使用 Zod 转换 TypeScript 类型声明

如下所示，SerpAPI 工具中对参数接口进行了详细的定义：

```
// Copied over from `serpapi` package
interface BaseParameters {
  /**
   * Parameter defines the device to use to get the results. It can be set to
   * `desktop` (default) to use a regular browser, `tablet` to use a tablet browser
   * (currently using iPads), or `mobile` to use a mobile browser (currently
   * using iPhones).
   */
  device?: "desktop" | "tablet" | "mobile";
  /**
   * Parameter will force SerpApi to fetch the Google results even if a cached
   * version is already present. A cache is served only if the query and all
   * parameters are exactly the same. Cache expires after 1h. Cached searches
   * are free, and are not counted towards your searches per month. It can be set
   * to `false` (default) to allow results from the cache, or `true` to disallow
   * results from the cache. `no_cache` and `async` parameters should not be used together.
   */
  no_cache?: boolean;
  /**
   * Specify the client-side timeout of the request. In milliseconds.
   */
```

Agent 工具箱中的工具集

工具集（Toolkit）是旨在一起用于特定任务并具有方便的加载方法的工具的集合

```
const toolkit = new JsonToolkit(new JsonSpec(data));
const model = new OpenAI({ temperature: 0 });
const executor = createJsonAgent(model, toolkit);
```

值得注意的是，为了更好地串联工具，工具集一般会重新定义 ReAct Agent 的部分提示词

```
export const JSON_PREFIX = `You are an agent designed to interact with JSON.
Your goal is to return a final answer by interacting with the JSON.
You have access to the following tools which help you learn more about the JSON you are interacting with.
Only use the below tools. Only use the information returned by the below tools to construct your final answer.
Do not make up any information that is not contained in the JSON.
Your input to the tools should be in the form of in json pointer syntax (e.g. /key1/0/key2).
You must escape a slash in a key with a ~1, and escape a tilde with a ~0.
For example, to access the key /foo, you would use /~1foo
You should only use keys that you know for a fact exist. You must validate that a key exists by seeing it previously whe
If you have not seen a key in one of those responses, you cannot use it.
You should only add one key at a time to the path. You cannot add multiple keys at once.
If you encounter a null or undefined value, go back to the previous key, look at the available keys, and try again.

If the question does not seem to be related to the JSON, just return "I don't know" as the answer.
```

Run: AgentExecutor

Playground

Trace

- AgentExecutor • Success
  - Chain LLMChain
  - Tool JsonListKeysTool
  - Chain LLMChain
  - Tool JsonListKeysTool
  - Chain LLMChain
  - Tool JsonListKeysTool
  - Chain LLMChain
  - Tool JsonListKeysTool
  - Chain LLMChain
  - Tool JsonListKeysTool
  - Chain LLMChain

AgentExecutor

Run | Feedback | Metadata

INPUT Copy

```yaml
input: What are the required parameters in the request body to the /completions endpoint?
```

YAML

OUTPUT Copy

```yaml
output: The required parameters for the /completions endpoint are "model".
intermediateSteps:
  - action:
      log: |-
```

Web API 调用： OpenAPI Agent

如下所示，我们会通过 OpenAPI Agent 来调用 OpenAI 的文本补全接口（参见 完整接口文件）

```
const headers = {
  "Content-Type": "application/json",
  Authorization: `Bearer ${process.env.OPENAI_API_KEY}`,
};
const model = new OpenAI({ temperature: 0 });
const toolkit = new OpenApiToolkit(new JsonSpec(data), model, headers);
const executor = createOpenApiAgent(model, toolkit);
```

```
export const OPENAPI_PREFIX = `You are an agent designed to answer questions by making web requests to an API given the

If the question does not seem related to the API, return I don't know. Do not make up an answer.
Only use information provided by the tools to construct your response.

To find information in the OpenAPI spec, use the 'json_explorer' tool. The input to this tool is a question about the AP

Take the following steps:
First, find the base URL needed to make the request.

Second, find the relevant paths needed to answer the question. Take note that, sometimes, you might need to make more th

Third, find the required parameters needed to make the request. For GET requests, these are usually URL parameters and t
```

Run: AgentExecutor

Playground

Trace

- AgentExecutor • Success
  - Chain LLMChain
    - LLM OpenAI
  - Tool DynamicTool
  - Chain LLMChain
    - LLM OpenAI
  - Tool DynamicTool
  - Chain LLMChain
    - LLM OpenAI
  - Tool DynamicTool
  - Chain LLMChain
    - LLM OpenAI

AgentExecutor

Run | Feedback | Metadata

INPUT Copy

```
1  input: Make a POST request to openai
   /completions. The prompt should be 'tell me a
   joke.'
```

YAML

OUTPUT Copy

```
1  output: "\".\\\"\\n\\n\\\"We're not the ones
   making all the jokes,\\\" Tam Jae\""
2  intermediateSteps:
3    - action:
```

Run: AgentExecutor

Playground

Trace

- AgentExecutor • Success
  - Chain LLMChain
    - LLM OpenAI
  - Tool JsonListKeysTool
  - Chain LLMChain
    - LLM OpenAI
  - Tool JsonListKeysTool
  - Chain LLMChain
    - LLM OpenAI
  - Tool JsonGetValueTool
  - Chain LLMChain
    - LLM OpenAI

AgentExecutor

Run | Feedback | Metadata

INPUT Copy

```
1 input: What is the base url for the API?
```

YAML

OUTPUT Copy

```
1 output: The base url for the API is
  https://api.openai.com/v1.
2 intermediateSteps:
3   - action:
4       log: |-
5         Action: json_list_keys
```

Run: AgentExecutor

Playground

Trace

- AgentExecutor • Success
  - Chain LLMChain
    - LLM OpenAI
  - Tool JsonListKeysTool
  - Chain LLMChain
    - LLM OpenAI
  - Tool JsonListKeysTool
  - Chain LLMChain
    - LLM OpenAI

AgentExecutor

Run | Feedback | Metadata

INPUT Copy

```
1  input: What is the path for the /completions
   endpoint?
```

YAML

OUTPUT Copy

```
1  output: The path for the /completions
   endpoint is /completions.
2  intermediateSteps:
3    - action:
4        log: |-
```

Run: AgentExecutor

Playground

Trace

- AgentExecutor • Success
  - Chain LLMChain
  - Tool JsonListKeysTool
  - Chain LLMChain
  - Tool JsonListKeysTool
  - Chain LLMChain
  - Tool JsonListKeysTool
  - Chain LLMChain
  - Tool JsonListKeysTool
  - Chain LLMChain
  - Tool JsonListKeysTool
  - Chain LLMChain

AgentExecutor

Run | Feedback | Metadata

INPUT Copy

```
1  input: What are the required parameters for a
   POST request to the /completions endpoint?
```

YAML

OUTPUT Copy

```
1  output: The required parameters for a POST
   request to the /completions endpoint are
   "model".
2  intermediateSteps:
3    - action:
```

向量存储也可以作用工具（集）

```
import { OpenAI } from "langchain/llms/openai";
import { HNSWLib } from "langchain/vectorstores/hnswlib";
import { OpenAIEmbeddings } from "langchain/embeddings/openai";
import { RecursiveCharacterTextSplitter } from "langchain/text_splitter";
import { VectorStoreToolkit, createVectorStoreAgent, VectorStoreInfo } from "langchain/agents";

const model = new OpenAI({ temperature: 0 });
/* Load in the file we want to do question answering over */
const text = fs.readFileSync("state_of_the_union.txt", "utf8");
/* Split the text into chunks using character, not token, size */
const textSplitter = new RecursiveCharacterTextSplitter({ chunkSize: 1000 });
const docs = await textSplitter.createDocuments([text]);
/* Create the vectorstore */
const vectorStore = await HNSWLib.fromDocuments(docs, new OpenAIEmbeddings());

/* Create the agent */
const vectorStoreInfo: VectorStoreInfo = {
  name: "state_of_union_address",
  description: "the most recent state of the Union address",
  vectorStore,
};

const toolkit = new VectorStoreToolkit(vectorStoreInfo, model);
```

Run: AgentExecutor

Playground

Trace

- AgentExecutor • Success
  - Chain LLMChain
    - LLM OpenAI
  - Tool VectorStoreQATool
  - Chain LLMChain
    - LLM OpenAI

AgentExecutor

Run | Feedback | Metadata

INPUT Copy

```
1 input: What did biden say about Ketanji Brown Jackson is the state of the union address?
```

YAML

OUTPUT Copy

```
1 output: Biden said that Jackson is one of the nation's top legal minds and will continue Justice Breyer's legacy of excellence. She is a former top litigator in private practice, a former federal public
```

必不可缺的 Callback 回调系统

Callback 回调系统让我们可以连接到 LLM 应用的各个阶段，这对于日志记录、监控、流传输等非常有用

`callbacks` 参数在整个 API（Model、Chain、Agent、Tool 等）的大多数对象上的两个不同位置可用

- 可以在构造器位置上接入 Callback（但不能跨对象使用），以 JS/TS 中的 Model 为例：

```
import { ConsoleCallbackHandler } from "langchain/callbacks";

const llm = new OpenAI({
  // These tags will be attached to all calls made with this LLM.
  tags: ["example", "callbacks", "constructor"],
  // This handler will be used for all calls made with this LLM.
  callbacks: [new ConsoleCallbackHandler()],
});
```

- 也可以在模块对象的 `apply()` / `run()` / `call()` 实例方法中发起请求时绑定，但仅对该请求有效：

```
const llm = new OpenAI({ temperature: 0 });
const response = await llm.call("1 + 1 =", {
  // These tags will be attached only to this call to the LLM.
  tags: ["example", "callbacks", "request"],
  // This handler will be used only for this call.
  callbacks: [new ConsoleCallbackHandler()],
});
```

回顾：链路调试的“文韬武略”

我们既可以接入强力的 LangSmith，也可以使用 Console 输出调试信息

如何获得足够多且跨模块对象 Console 调试信息 ➡️ 开启 `verbose` 选项：

```
export LANGCHAIN_VERBOSE=true
```

```
const executor = await initializeAgentExecutorWithOptions(tools, model, { agentType: "xml", verbose: true });
```

或者直接通过环境变量接入 LangSmith：

```
export LANGCHAIN_TRACING_V2=true
export LANGCHAIN_ENDPOINT=https://api.smith.langchain.com
export LANGCHAIN_API_KEY=<your-api-key>  # still in closed beta
export LANGCHAIN_PROJECT=<your-project>  # if not specified, defaults to "default"
```

⚠ 以下方式只能用于对象构造器和对象请求，不能形成完整链路：

```
import { Client } from "langsmith";
import { LangChainTracer } from "langchain/callbacks";

const client = new Client({ apiUrl: "https://api.smith.langchain.com", apiKey: "YOUR_API_KEY" });
const tracer = new LangChainTracer({ projectName: "YOUR_PROJECT_NAME", client });
```

如何自定义 Callback 处理器

LangChain 支持通过实现基本回调处理程序接口来创建您自己的处理程序

自定义 Callback Handler 可以做一些比输出调试信息到控制台更复杂的事情，例如将事件发送到日志记录服务。作为示例，这里是一个记录到控制台的处理程序的 JS/TS 版本简单实现：

```
import { AgentAction, AgentFinish, ChainValues } from "langchain/schema";

export class MyCallbackHandler extends BaseCallbackHandler {
  name = "MyCallbackHandler";

  async handleChainStart(chain: Serialized) {
    console.log(`Entering new ${chain.id} chain...`);
  }

  async handleChainEnd(_output: ChainValues) {
    console.log("Finished chain.");
  }

  async handleAgentAction(action: AgentAction) {
    console.log(action.log);
  }

  async handleToolEnd(output: string) {
    console.log(output);
```

参考资料

本教程在制作过程中参考和引用了以下资料（排名不分先后）的内容，特此鸣谢！

视频资料

- Short Courses | Learn Generative AI from DeepLearning.AI

图文资料

- Core Concepts | 🦜🔗 LangChain
  - JS/TS Docs, Python Docs, LangSmith Docs
- 入门：Prompts（提示词）| 通往 AGI 之路

代码资料

- openai/openai-cookbook: Examples and guides for using the OpenAI API
- datawhalechina/prompt-engineering-for-developers: 吴恩达大模型系列课程中文版
- slidevjs/slidev: Presentation Slides for Developers

感谢聆听 ❤️

webup | serviceup