# 代理类型

尽管现代的语言模型，如GPT-4，已经极其强大，能够理解和生成高度复杂和连贯的文本，然而，这些模型往往还无法独立完成特定的任务。这就是代理的作用。代理是一种灵活的接口，允许语言模型和其他工具之间形成一个灵活的调用链，来解决特定的问题或任务。这种灵活性，使得语言模型可以适应更多不同的应用场景，实现其单独无法完成的功能。

具体来说，代理拥有一整套工具的访问权限，它可以根据用户的输入来决定使用哪些工具。一个代理可以使用多种工具，甚至可以把一个工具的输出作为下一个工具的输入。通过这种方式，代理将工具和语言模型有机地结合在一起，实现了高度复杂和特定的任务。

根据任务的不同，代理主要有两种类型：行动代理和计划执行代理。

行动代理：每个时间步都会根据前面所有动作的输出来决定下一步的行动。行动代理适合于小型任务，它的优点在于能够实时地处理信息和作出决策。

计划执行代理：首先决定全部行动序列，然后一次性执行所有的动作，而不更新计划。计划执行代理更适合于需要保持长期目标和重点的复杂或长期运行的任务。

值得注意的是，常常将这两种代理结合起来使用是最佳的做法，也就是说，让计划执行代理使用行动代理来执行计划。这样的结合，既保持了行动代理的动态性，又利用了计划执行代理的规划能力，从而最大化地发挥了代理的功能和效用。

#### 行动代理

代理利用语言大模型（LLM）决定要执行哪些行动以及行动的顺序。行动可以是使用工具并观察其输出，也可以是向用户返回响应。

首先，让我们查看源码里代理的类型。

class AgentType(str, Enum):  
 ZERO\_SHOT\_REACT\_DESCRIPTION = "zero-shot-react-description"  
 REACT\_DOCSTORE = "react-docstore"  
 SELF\_ASK\_WITH\_SEARCH = "self-ask-with-search"  
 CONVERSATIONAL\_REACT\_DESCRIPTION = "conversational-react-description"  
 CHAT\_ZERO\_SHOT\_REACT\_DESCRIPTION = "chat-zero-shot-react-description"  
 CHAT\_CONVERSATIONAL\_REACT\_DESCRIPTION = "chat-conversational-react-description"  
 STRUCTURED\_CHAT\_ZERO\_SHOT\_REACT\_DESCRIPTION = (  
 "structured-chat-zero-shot-react-description"  
 )  
 OPENAI\_FUNCTIONS = "openai-functions"

1. Zero-shot ReAct 这种代理使用ReAct框架，仅根据工具的描述决定使用哪个工具。可以提供任意数量的工具。这种代理要求为每个工具提供一个描述。
2. 结构化输入反应（Structured input ReAct） 结构化工具聊天代理能够使用多输入工具。旧代理被配置为将行动输入指定为单个字符串，但是这种代理可以使用工具的参数模式创建结构化的行动输入。这对于更复杂的工具使用，例如精确导航浏览器，非常有用。
3. OpenAI函数（OpenAI Functions） 某些OpenAI模型（如gpt-3.5-turbo-0613和gpt-4-0613）被明确地微调，以便检测何时应该调用函数，并响应应传递给函数的输入。OpenAI函数代理旨在与这些模型一起工作。
4. 对话（Conversational） 这种代理被设计用于对话设置。提示被设计为使代理有帮助和交谈。它使用ReAct框架决定使用哪个工具，并使用记忆来记住以前的对话交互。
5. 自问与搜索（Self ask with search） 这种代理使用一个名为中间答案的工具。这个工具应该能够查找问题的事实答案。这个代理相当于原始的self ask with search paper论文，其中提供了Google搜索API作为工具。
6. ReAct文档存储（ReAct document store） 这种代理使用ReAct框架与文档存储进行交互。必须提供两个工具：搜索工具和查找工具（必须精确地命名为这样）。搜索工具应该搜索一个文档，而查找工具应该在最近找到的文档中查找一个词条。这个代理等同于原始的ReAct论文，特别是维基百科的例子。

#### 计划并执行代理

紧接着行动代理的类型，让我们继续了解计划并执行代理的类型。

计划并执行代理（Plan-and-Solve Agents）通过首先规划要做什么，然后执行子任务来实现目标。这个思想很大程度上受到BabyAGI以及”Plan-and-Solve”论文的启发。

每种代理类型都有其特定的用途和应用场景，代理的灵活性和丰富性为LangChain提供了强大的功能性。