第七章 代理

大型语言模型（LLMs）非常强大，但它们缺乏“最笨”的计算机程序可以轻松处理的特定能力。LLM 对逻辑推理、计算和检索外部信息的能力较弱，这与最简单的计算机程序形成对比。例如，语言模型无法准确回答简单的计算问题，还有当询问最近发生的事件时，其回答也可能过时或错误，因为无法主动获取最新信息。这是由于当前语言模型仅依赖预训练数据，与外界“断开”。要克服这一缺陷，`LangChain` 框架提出了 `“代理”(Agent)` 的解决方案。

代理作为语言模型的外部模块，可提供计算、逻辑、检索等功能的支持，使语言模型获得异常强大的推理和获取信息的超能力。

在本章中，我们将详细介绍代理的工作机制、种类、以及如何在 `LangChain` 中将其与语言模型配合，构建功能更全面、智能程度更高的应用程序。代理机制极大扩展了语言模型的边界，是当前提升其智能的重要途径之一。让我们开始学习如何通过代理释放语言模型的最大潜力。

一、使用LangChain内置工具llm-math和wikipedia

要使用代理 (Agents) ，我们需要三样东西：

- 一个基本的 LLM
- 我们将要进行交互的工具 Tools
- 一个控制交互的代理 (Agents) 。

```python
from langchain.agents import load_tools, initialize_agent
from langchain.agents import AgentType
from langchain.python import PythonREPL
from langchain.chat_models import ChatOpenAI
```

首先，让我们新建一个基本的 LLM

```python
# 参数temperature设置为0.0，从而减少生成答案的随机性。
llm = ChatOpenAI(temperature=0)
```

接下来，初始化 `工具 Tool` ，我们可以创建自定义工具 Tool 或加载预构建工具 Tool。无论哪种情况，工具 Tool 都是一个给定工具 `名称 name` 和 `描述 description` 的 实用链。

- `llm-math` 工具结合语言模型和计算器用以进行数学计算
- `wikipedia` 工具通过API连接到wikipedia进行搜索查询。

```python

tools = load_tools(
    ["llm-math","wikipedia"],
    llm=llm #第一步初始化的模型
)
```

现在我们有了 LLM 和工具，最后让我们初始化一个简单的代理 (Agents) ：

```
# 初始化代理
agent= initialize_agent(
    tools, #第二步加载的工具
    llm, #第一步初始化的模型
    agent=AgentType.CHAT_ZERO_SHOT_REACT_DESCRIPTION,  #代理类型
    handle_parsing_errors=True, #处理解析错误
    verbose = True #输出中间步骤
)
```

- `agent`: 代理类型。这里使用的是`AgentType.CHAT_ZERO_SHOT_REACT_DESCRIPTION`。其中`CHAT`代表代理模型为针对对话优化的模型；`Zero-shot` 意味着代理 (Agents) 仅在当前操作上起作用，即它没有记忆；`REACT`代表针对REACT设计的提示模版。`DESCRIPTION`根据工具的描述 description 来决定使用哪个工具。(我们不会在本章中讨论 * REACT 框架 ，但您可以将其视为 LLM 可以循环进行 Reasoning 和 Action 步骤的过程。它启用了一个多步骤的过程来识别答案。)
- `handle_parsing_errors`: 是否处理解析错误。当发生解析错误时，将错误信息返回给大模型，让其进行纠正。
- `verbose`: 是否输出中间步骤结果。

使用代理回答数学问题

```
agent("计算300的25%")
```

~~~
> Entering new AgentExecutor chain...
Question: 计算300的25%
Thought: I can use the calculator tool to calculate 25% of 300.
Action:
```json
{
  "action": "Calculator",
  "action_input": "300 * 0.25"
}
```


Observation: Answer: 75.0
Thought:The calculator tool returned the answer 75.0, which is 25% of 300.
Final Answer: 25% of 300 is 75.0.

> Finished chain.
~~~

```
{'input': '计算300的25%', 'output': '25% of 300 is 75.0.'}
```

上面的过程可以总结为下

1. 模型对于接下来需要做什么，给出思考

   思考：我可以使用计算工具来计算300的25%

2. 模型基于思考采取行动

   行动：使用计算器（calculator），输入（action_input）300*0.25

3. 模型得到观察

   观察：答案：75.0

4. 基于观察，模型对于接下来需要做什么，给出思考

   思考: 计算工具返回了300的25%，答案为75

5. 给出最终答案（Final Answer）

   最终答案: 300的25%等于75。

6. 以字典的形式给出最终答案。

Tom M. Mitchell的书

```
question = "Tom M. Mitchell是一位美国计算机科学家，\
也是卡内基梅隆大学（CMU）的创始人大学教授。\
他写了哪本书呢？"

agent(question)
```

~~~
> Entering new AgentExecutor chain...
Thought: I can use Wikipedia to find information about Tom M. Mitchell and his
books.
Action:
```json
{
  "action": "Wikipedia",
  "action_input": "Tom M. Mitchell"
}
```
Observation: Page: Tom M. Mitchell
Summary: Tom Michael Mitchell (born August 9, 1951) is an American computer
scientist and the Founders University Professor at Carnegie Mellon University
(CMU). He is a founder and former Chair of the Machine Learning Department at
CMU. Mitchell is known for his contributions to the advancement of machine
learning, artificial intelligence, and cognitive neuroscience and is the author
of the textbook Machine Learning. He is a member of the United States National
Academy of Engineering since 2010. He is also a Fellow of the American Academy of
Arts and Sciences, the American Association for the Advancement of Science and a
Fellow and past President of the Association for the Advancement of Artificial
Intelligence. In October 2018, Mitchell was appointed as the Interim Dean of the
School of Computer Science at Carnegie Mellon.

Page: Tom Mitchell (Australian footballer)
Summary: Thomas Mitchell (born 31 May 1993) is a professional Australian rules
footballer playing for the Collingwood Football Club in the Australian Football
League (AFL). He previously played for the Adelaide Crows, Sydney Swans from 2012
to 2016, and the Hawthorn Football Club between 2017 and 2022. Mitchell won the
Brownlow Medal as the league's best and fairest player in 2018 and set the record
for the most disposals in a VFL/AFL match, accruing 54 in a game against
Collingwood during that season.
Thought:The book written by Tom M. Mitchell is "Machine Learning".
Thought: I have found the answer.
Final Answer: The book written by Tom M. Mitchell is "Machine Learning".

> Finished chain.
~~~

```
{'input': 'Tom M. Mitchell是一位美国计算机科学家，也是卡内基梅隆大学（CMU）的创始人大学教授。他写了哪本书呢？',
 'output': 'The book written by Tom M. Mitchell is "Machine Learning".'}
```

✅ **总结**

1. 模型对于接下来需要做什么，给出思考（Thought）

   思考：我应该使用维基百科去搜索。

2. 模型基于思考采取行动（Action）

   行动：使用维基百科，输入Tom M. Mitchell

3. 模型得到观察（Observation）

   观测：页面：Tom M. Mitchell，页面：Tom Mitchell (澳大利亚足球运动员)

4. 基于观察，模型对于接下来需要做什么，给出思考（Thought）

   思考：Tom M. Mitchell写的书是Machine Learning

5. 给出最终答案（Final Answer）

   最终答案：Machine Learning

6. 以字典的形式给出最终答案。

值得注意的是，模型每次运行推理的过程可能存在差异，但最终的结果一致。

二、 使用LangChain内置工具PythonREPLTool

我们创建一个能将顾客名字转换为拼音的 python 代理，步骤与上一部分的一样：

```python
from langchain.agents.agent_toolkits import create_python_agent
from langchain.tools.python.tool import PythonREPLTool

agent = create_python_agent(
    llm,  #使用前面一节已经加载的大语言模型
    tool=PythonREPLTool(), #使用Python交互式环境工具 REPLTool
    verbose=True #输出中间步骤
)
customer_list = ["小明","小黄","小红","小蓝","小橘","小绿",]

agent.run(f"将使用pinyin拼音库这些客户名字转换为拼音，并打印输出列表: {customer_list}。")
```

```
> Entering new AgentExecutor chain...
I need to use the pinyin library to convert the names to pinyin. I can then print
out the list of converted names.
Action: Python_REPL
Action Input: import pinyin
Observation:
Thought:I have imported the pinyin library. Now I can use it to convert the names
to pinyin.
Action: Python_REPL
Action Input: names = ['小明', '小黄', '小红', '小蓝', '小橘', '小绿']
pinyin_names = [pinyin.get(i, format='strip') for i in names]
print(pinyin_names)
Observation: ['xiaoming', 'xiaohuang', 'xiaohong', 'xiaolan', 'xiaoju', 'xiaolv']
```

```
Thought:I have successfully converted the names to pinyin and printed out the
list of converted names.
Final Answer: ['xiaoming', 'xiaohuang', 'xiaohong', 'xiaolan', 'xiaoju',
'xiaolv']

> Finished chain.
```

```
"['xiaoming', 'xiaohuang', 'xiaohong', 'xiaolan', 'xiaoju', 'xiaolv']"
```

在调试（debug）模式下再次运行，我们可以把上面的6步分别对应到下面的具体流程

1. 模型对于接下来需要做什么，给出思考（Thought）
    - [chain/start] [1:chain:AgentExecutor] Entering Chain run with input
    - [chain/start] [1:chain:AgentExecutor > 2:chain:LLMChain] Entering Chain run with input
    - [llm/start] [1:chain:AgentExecutor > 2:chain:LLMChain > 3:llm:ChatOpenAI] Entering LLM run with input
    - [llm/end] [1:chain:AgentExecutor > 2:chain:LLMChain > 3:llm:ChatOpenAI] [1.91s] Exiting LLM run with output
    - [chain/end] [1:chain:AgentExecutor > 2:chain:LLMChain] [1.91s] Exiting Chain run with output
2. 模型基于思考采取行动（Action), 因为使用的工具不同，Action的输出也和之前有所不同，这里输出的为python代码 `import pinyin`
    - [tool/start] [1:chain:AgentExecutor > 4:tool:Python REPL] Entering Tool run with input
    - [tool/end] [1:chain:AgentExecutor > 4:tool:Python_REPL] [1.28ms] Exiting Tool run with output
3. 模型得到观察（Observation）
    - [chain/start] [1:chain:AgentExecutor > 5:chain:LLMChain] Entering Chain run with input
4. 基于观察，模型对于接下来需要做什么，给出思考（Thought）
    - [llm/start] [1:chain:AgentExecutor > 5:chain:LLMChain > 6:llm:ChatOpenAI] Entering LLM run with input
    - [llm/end] [1:chain:AgentExecutor > 5:chain:LLMChain > 6:llm:ChatOpenAI] [3.48s] Exiting LLM run with output
5. 给出最终答案（Final Answer）
    - [chain/end] [1:chain:AgentExecutor > 5:chain:LLMChain] [3.48s] Exiting Chain run with output
6. 返回最终答案。
    - [chain/end] [1:chain:AgentExecutor] [19.20s] Exiting Chain run with output

```python
import langchain
langchain.debug=True
agent.run(f"使用pinyin拼音库将这些客户名字转换为拼音，并打印输出列表: {customer_list}")
langchain.debug=False
```

```
[chain/start] [1:chain:AgentExecutor] Entering Chain run with input:
```

```
{
  "input": "使用pinyin拼音库将这些客户名字转换为拼音，并打印输出列表: ['小明', '小黄', '小红', '小蓝', '小橘', '小绿']"
}
[chain/start] [1:chain:AgentExecutor > 2:chain:LLMChain] Entering Chain run with input:
{
  "input": "使用pinyin拼音库将这些客户名字转换为拼音，并打印输出列表: ['小明', '小黄', '小红', '小蓝', '小橘', '小绿']",
  "agent_scratchpad": "",
  "stop": [
    "\nObservation:",
    "\n\tObservation:"
  ]
}
[llm/start] [1:chain:AgentExecutor > 2:chain:LLMChain > 3:llm:ChatOpenAI] Entering LLM run with input:
{
  "prompts": [
    "Human: You are an agent designed to write and execute python code to answer questions.\nYou have access to a python REPL, which you can use to execute python code.\nIf you get an error, debug your code and try again.\nOnly use the output of your code to answer the question. \nYou might know the answer without running any code, but you should still run the code to get the answer.\nIf it does not seem like you can write code to answer the question, just return \"I don't know\" as the answer.\n\n\nPython_REPL: A Python shell. Use this to execute python commands. Input should be a valid python command. If you want to see the output of a value, you should print it out with `print(...)`.\n\nUse the following format:\n\nQuestion: the input question you must answer\nThought: you should always think about what to do\nAction: the action to take, should be one of [Python_REPL]\nAction Input: the input to the action\nObservation: the result of the action\n... (this Thought/Action/Action Input/Observation can repeat N times)\nThought: I now know the final answer\nFinal Answer: the final answer to the original input question\n\nBegin!\n\nQuestion: 使用pinyin拼音库将这些客户名字转换为拼音，并打印输出列表: ['小明', '小黄', '小红', '小蓝', '小橘', '小绿']\nThought:"
  ]
}
[llm/end] [1:chain:AgentExecutor > 2:chain:LLMChain > 3:llm:ChatOpenAI] [2.32s] Exiting LLM run with output:
{
  "generations": [
    [
      {
        "text": "I need to use the pinyin library to convert the names to pinyin. I can then print out the list of converted names.\nAction: Python_REPL\nAction Input: import pinyin",
        "generation_info": {
          "finish_reason": "stop"
        },
        "message": {
          "lc": 1,
          "type": "constructor",
          "id": [
            "langchain",
            "schema",
            "messages",
            "AIMessage"
          ],
          "kwargs": {
            "content": "I need to use the pinyin library to convert the names to
pinyin. I can then print out the list of converted names.\nAction:
Python_REPL\nAction Input: import pinyin",
            "additional_kwargs": {}
          }
        }
      }
    ]
  ],
  "llm_output": {
    "token_usage": {
      "prompt_tokens": 320,
      "completion_tokens": 39,
      "total_tokens": 359
    },
    "model_name": "gpt-3.5-turbo"
  },
  "run": null
}
[chain/end] [1:chain:AgentExecutor > 2:chain:LLMChain] [2.33s] Exiting Chain run
with output:
{
  "text": "I need to use the pinyin library to convert the names to pinyin. I can
then print out the list of converted names.\nAction: Python_REPL\nAction Input:
import pinyin"
}
[tool/start] [1:chain:AgentExecutor > 4:tool:Python_REPL] Entering Tool run with
input:
"import pinyin"
[tool/end] [1:chain:AgentExecutor > 4:tool:Python_REPL] [1.5659999999999998ms]
Exiting Tool run with output:
""
[chain/start] [1:chain:AgentExecutor > 5:chain:LLMChain] Entering Chain run with
input:
{
  "input": "使用pinyin拼音库将这些客户名字转换为拼音，并打印输出列表: ['小明', '小黄', '小
红', '小蓝', '小橘', '小绿']",
  "agent_scratchpad": "I need to use the pinyin library to convert the names to
pinyin. I can then print out the list of converted names.\nAction:
Python_REPL\nAction Input: import pinyin\nObservation: \nThought:",
  "stop": [
    "\nObservation:",
    "\n\tObservation:"
  ]
}
[llm/start] [1:chain:AgentExecutor > 5:chain:LLMChain > 6:llm:ChatOpenAI]
Entering LLM run with input:
{
  "prompts": [
    "Human: You are an agent designed to write and execute python code to answer questions.\nYou have access to a python REPL, which you can use to execute python code.\nIf you get an error, debug your code and try again.\nOnly use the output of your code to answer the question. \nYou might know the answer without running any code, but you should still run the code to get the answer.\nIf it does not seem like you can write code to answer the question, just return \"I don't know\" as the answer.\n\n\nPython_REPL: A Python shell. Use this to execute python commands. Input should be a valid python command. If you want to see the output of a value, you should print it out with `print(...)`.\n\nUse the following format:\n\nQuestion: the input question you must answer\nThought: you should always think about what to do\nAction: the action to take, should be one of [Python_REPL]\nAction Input: the input to the action\nObservation: the result of the action\n... (this Thought/Action/Action Input/Observation can repeat N times)\nThought: I now know the final answer\nFinal Answer: the final answer to the original input question\n\nBegin!\n\nQuestion: 使用pinyin拼音库将这些客户名字转换为拼音，并打印输出列表: ['小明', '小黄', '小红', '小蓝', '小橘', '小绿']\nThought:I need to use the pinyin library to convert the names to pinyin. I can then print out the list of converted names.\nAction: Python_REPL\nAction Input: import pinyin\nObservation: \nThought:"
  ]
}
[llm/end] [1:chain:AgentExecutor > 5:chain:LLMChain > 6:llm:ChatOpenAI] [4.09s] Exiting LLM run with output:
{
  "generations": [
    [
      {
        "text": "I have imported the pinyin library. Now I can use it to convert the names to pinyin.\nAction: Python_REPL\nAction Input: names = ['小明', '小黄', '小红', '小蓝', '小橘', '小绿']\npinyin_names = [pinyin.get(i, format='strip') for i in names]\nprint(pinyin_names)",
        "generation_info": {
          "finish_reason": "stop"
        },
        "message": {
          "lc": 1,
          "type": "constructor",
          "id": [
            "langchain",
            "schema",
            "messages",
            "AIMessage"
          ],
          "kwargs": {
            "content": "I have imported the pinyin library. Now I can use it to convert the names to pinyin.\nAction: Python_REPL\nAction Input: names = ['小明', '小黄', '小红', '小蓝', '小橘', '小绿']\npinyin_names = [pinyin.get(i, format='strip') for i in names]\nprint(pinyin_names)",
            "additional_kwargs": {}
          }
        }
      }
    ]
  ],
  "llm_output": {
    "token_usage": {
      "prompt_tokens": 365,
      "completion_tokens": 87,
      "total_tokens": 452
    },
    "model_name": "gpt-3.5-turbo"
  },
  "run": null
}
[chain/end] [1:chain:AgentExecutor > 5:chain:LLMChain] [4.09s] Exiting Chain run
with output:
{
  "text": "I have imported the pinyin library. Now I can use it to convert the
names to pinyin.\nAction: Python_REPL\nAction Input: names = ['小明', '小黄', '小
红', '小蓝', '小橘', '小绿']\npinyin_names = [pinyin.get(i, format='strip') for i
in names]\nprint(pinyin_names)"
}
[tool/start] [1:chain:AgentExecutor > 7:tool:Python_REPL] Entering Tool run with
input:
"names = ['小明', '小黄', '小红', '小蓝', '小橘', '小绿']
pinyin_names = [pinyin.get(i, format='strip') for i in names]
print(pinyin_names)"
[tool/end] [1:chain:AgentExecutor > 7:tool:Python_REPL] [0.8809999999999999ms]
Exiting Tool run with output:
"['xiaoming', 'xiaohuang', 'xiaohong', 'xiaolan', 'xiaoju', 'xiaolv']"
[chain/start] [1:chain:AgentExecutor > 8:chain:LLMChain] Entering Chain run with
input:
{
  "input": "使用pinyin拼音库将这些客户名字转换为拼音，并打印输出列表: ['小明', '小黄', '小
红', '小蓝', '小橘', '小绿']",
  "agent_scratchpad": "I need to use the pinyin library to convert the names to
pinyin. I can then print out the list of converted names.\nAction:
Python_REPL\nAction Input: import pinyin\nObservation: \nThought:I have imported
the pinyin library. Now I can use it to convert the names to pinyin.\nAction:
Python_REPL\nAction Input: names = ['小明', '小黄', '小红', '小蓝', '小橘', '小
绿']\npinyin_names = [pinyin.get(i, format='strip') for i in
names]\nprint(pinyin_names)\nObservation: ['xiaoming', 'xiaohuang', 'xiaohong',
'xiaolan', 'xiaoju', 'xiaolv']\n\nThought:",
  "stop": [
    "\nObservation:",
    "\n\tObservation:"
  ]
}
[llm/start] [1:chain:AgentExecutor > 8:chain:LLMChain > 9:llm:ChatOpenAI]
Entering LLM run with input:
{
  "prompts": [
    "Human: You are an agent designed to write and execute python code to answer questions.\nYou have access to a python REPL, which you can use to execute python code.\nIf you get an error, debug your code and try again.\nOnly use the output of your code to answer the question. \nYou might know the answer without running any code, but you should still run the code to get the answer.\nIf it does not seem like you can write code to answer the question, just return \"I don't know\" as the answer.\n\n\nPython_REPL: A Python shell. Use this to execute python commands. Input should be a valid python command. If you want to see the output of a value, you should print it out with `print(...)`.\n\nUse the following format:\n\nQuestion: the input question you must answer\nThought: you should always think about what to do\nAction: the action to take, should be one of [Python_REPL]\nAction Input: the input to the action\nObservation: the result of the action\n... (this Thought/Action/Action Input/Observation can repeat N times)\nThought: I now know the final answer\nFinal Answer: the final answer to the original input question\n\nBegin!\n\nQuestion: 使用pinyin拼音库将这些客户名字转换为拼音，并打印输出列表: ['小明', '小黄', '小红', '小蓝', '小橘', '小绿']\nThought:I need to use the pinyin library to convert the names to pinyin. I can then print out the list of converted names.\nAction: Python_REPL\nAction Input: import pinyin\nObservation: \nThought:I have imported the pinyin library. Now I can use it to convert the names to pinyin.\nAction: Python_REPL\nAction Input: names = ['小明', '小黄', '小红', '小蓝', '小橘', '小绿']\npinyin_names = [pinyin.get(i, format='strip') for i in names]\nprint(pinyin_names)\nObservation: ['xiaoming', 'xiaohuang', 'xiaohong', 'xiaolan', 'xiaoju', 'xiaolv']\n\nThought:"
  ]
}
[llm/end] [1:chain:AgentExecutor > 8:chain:LLMChain > 9:llm:ChatOpenAI] [2.05s] Exiting LLM run with output:
{
  "generations": [
    [
      {
        "text": "I have successfully converted the names to pinyin and printed out the list of converted names.\nFinal Answer: ['xiaoming', 'xiaohuang', 'xiaohong', 'xiaolan', 'xiaoju', 'xiaolv']",
        "generation_info": {
          "finish_reason": "stop"
        },
        "message": {
          "lc": 1,
          "type": "constructor",
          "id": [
            "langchain",
            "schema",
            "messages",
            "AIMessage"
          ],
          "kwargs": {
            "content": "I have successfully converted the names to pinyin and printed out the list of converted names.\nFinal Answer: ['xiaoming', 'xiaohuang', 'xiaohong', 'xiaolan', 'xiaoju', 'xiaolv']",
            "additional_kwargs": {}
          }
        }
      }
    ]
  ],
  "llm_output": {
    "token_usage": {
      "prompt_tokens": 483,
      "completion_tokens": 48,
      "total_tokens": 531
    },
    "model_name": "gpt-3.5-turbo"
  },
  "run": null
}
[chain/end] [1:chain:AgentExecutor > 8:chain:LLMChain] [2.05s] Exiting Chain run
with output:
{
  "text": "I have successfully converted the names to pinyin and printed out the
list of converted names.\nFinal Answer: ['xiaoming', 'xiaohuang', 'xiaohong',
'xiaolan', 'xiaoju', 'xiaolv']"
}
[chain/end] [1:chain:AgentExecutor] [8.47s] Exiting Chain run with output:
{
  "output": "['xiaoming', 'xiaohuang', 'xiaohong', 'xiaolan', 'xiaoju',
'xiaolv']"
}
```

三、 定义自己的工具并在代理中使用

在本节，我们将**创建和使用自定义时间工具。LangChian tool 函数装饰器可以应用用于任何函数，将函数转化为LangChain 工具，使其成为代理可调用的工具。**我们需要给函数加上非常详细的文档字符串，使得代理知道在什么情况下、如何使用该函数/工具。比如下面的函数`time`,我们加上了详细的文档字符串。

```python
# 导入tool函数装饰器
from langchain.agents import tool
from datetime import date

@tool
def time(text: str) -> str:
    """
    返回今天的日期，用于任何需要知道今天日期的问题。\
    输入应该总是一个空字符串，\
    这个函数将总是返回今天的日期，任何日期计算应该在这个函数之外进行。
    """
    return str(date.today())

# 初始化代理
agent= initialize_agent(
    tools=[time], #将刚刚创建的时间工具加入代理
    llm=llm, #初始化的模型
    agent=AgentType.CHAT_ZERO_SHOT_REACT_DESCRIPTION,  #代理类型
    handle_parsing_errors=True, #处理解析错误
    verbose = True #输出中间步骤
)

# 使用代理询问今天的日期.
# 注: 代理有时候可能会出错（该功能正在开发中）。如果出现错误，请尝试再次运行它。
```

```
agent("今天的日期是？")
```

```
> Entering new AgentExecutor chain...
根据提供的工具，我们可以使用`time`函数来获取今天的日期。

Thought: 使用`time`函数来获取今天的日期。

Action:
```
{
  "action": "time",
  "action_input": ""
}
```

Observation: 2023-08-09
Thought:我现在知道了最终答案。
Final Answer: 今天的日期是2023-08-09。

> Finished chain.
```

```
{'input': '今天的日期是？', 'output': '今天的日期是2023-08-09。'}
```

上面的过程可以总结为下

1. 模型对于接下来需要做什么，给出思考（Thought）

   思考：我需要使用 time 工具来获取今天的日期

2. 模型基于思考采取行动（Action), 因为使用的工具不同，Action的输出也和之前有所不同，这里输出的为python代码

   行动: 使用time工具，输入为空字符串

3. 模型得到观察（Observation）

   观测：2023-07-04

4. 基于观察，模型对于接下来需要做什么，给出思考（Thought）

   思考: 我已成功使用 time 工具检索到了今天的日期

5. 给出最终答案（Final Answer）

   最终答案: 今天的日期是2023-08-09.

6. 返回最终答案。

四、英文版

1. 使用LangChain内置工具llm-math和wikipedia

```python
from langchain.agents import load_tools, initialize_agent
from langchain.agents import AgentType
from langchain.python import PythonREPL
from langchain.chat_models import ChatOpenAI

llm = ChatOpenAI(temperature=0)
tools = load_tools(
    ["llm-math","wikipedia"],
    llm=llm
)


agent= initialize_agent(
    tools,
    llm,
    agent=AgentType.CHAT_ZERO_SHOT_REACT_DESCRIPTION,
    handle_parsing_errors=True,
    verbose = True
)

agent("What is the 25% of 300?")
```

~~~
> Entering new AgentExecutor chain...
I can use the calculator tool to find the answer to this question.

Action:
```json
{
  "action": "Calculator",
  "action_input": "25% of 300"
}
```
Observation: Answer: 75.0
Thought:The answer is 75.0.
Final Answer: 75.0

> Finished chain.
~~~

```
{'input': 'What is the 25% of 300?', 'output': '75.0'}
```

Tom M. Mitchell的书

```
from langchain.agents import load_tools, initialize_agent
from langchain.agents import AgentType
from langchain.python import PythonREPL
from langchain.chat_models import ChatOpenAI

llm = ChatOpenAI(temperature=0)
tools = load_tools(
    ["llm-math","wikipedia"],
    llm=llm
)


agent= initialize_agent(
    tools,
    llm,
    agent=AgentType.CHAT_ZERO_SHOT_REACT_DESCRIPTION,
    handle_parsing_errors=True,
    verbose = True
)

question = "Tom M. Mitchell is an American computer scientist \
```

```
and the Founders University Professor at Carnegie Mellon University (CMU)\
what book did he write?"
agent(question)
```

~~~
> Entering new AgentExecutor chain...
Thought: I can use Wikipedia to find out what book Tom M. Mitchell wrote.
Action:
```json
{
  "action": "Wikipedia",
  "action_input": "Tom M. Mitchell"
}
```
Observation: Page: Tom M. Mitchell
Summary: Tom Michael Mitchell (born August 9, 1951) is an American computer
scientist and the Founders University Professor at Carnegie Mellon University
(CMU). He is a founder and former Chair of the Machine Learning Department at
CMU. Mitchell is known for his contributions to the advancement of machine
learning, artificial intelligence, and cognitive neuroscience and is the author
of the textbook Machine Learning. He is a member of the United States National
Academy of Engineering since 2010. He is also a Fellow of the American Academy of
Arts and Sciences, the American Association for the Advancement of Science and a
Fellow and past President of the Association for the Advancement of Artificial
Intelligence. In October 2018, Mitchell was appointed as the Interim Dean of the
School of Computer Science at Carnegie Mellon.

Page: Tom Mitchell (Australian footballer)
Summary: Thomas Mitchell (born 31 May 1993) is a professional Australian rules
footballer playing for the Collingwood Football Club in the Australian Football
League (AFL). He previously played for the Adelaide Crows, Sydney Swans from 2012
to 2016, and the Hawthorn Football Club between 2017 and 2022. Mitchell won the
Brownlow Medal as the league's best and fairest player in 2018 and set the record
for the most disposals in a VFL/AFL match, accruing 54 in a game against
Collingwood during that season.
Thought:The book that Tom M. Mitchell wrote is "Machine Learning".
Final Answer: Machine Learning

> Finished chain.
~~~

```
{'input': 'Tom M. Mitchell is an American computer scientist and the Founders
University Professor at Carnegie Mellon University (CMU)what book did he write?',
 'output': 'Machine Learning'}
```

2. 使用LangChain内置工具PythonREPLTool

```
from langchain.agents.agent_toolkits import create_python_agent
from langchain.tools.python.tool import PythonREPLTool

agent = create_python_agent(
    llm,  #使用前面一节已经加载的大语言模型
    tool=PythonREPLTool(), #使用Python交互式环境工具（REPLTool）
    verbose=True #输出中间步骤
)
```

```
customer_list = [["Harrison", "Chase"],
                 ["Lang", "Chain"],
                 ["Dolly", "Too"],
                 ["Elle", "Elem"],
                 ["Geoff","Fusion"],
                 ["Trance","Former"],
                 ["Jen","Ayai"]
                ]
agent.run(f"""Sort these customers by \
last name and then first name \
and print the output: {customer_list}""")
```

```
> Entering new AgentExecutor chain...
I can use the `sorted()` function to sort the list of customers. I will need to
provide a key function that specifies the sorting order based on last name and
then first name.
Action: Python_REPL
Action Input: sorted([['Harrison', 'Chase'], ['Lang', 'Chain'], ['Dolly', 'Too'],
['Elle', 'Elem'], ['Geoff', 'Fusion'], ['Trance', 'Former'], ['Jen', 'Ayai']],
key=lambda x: (x[1], x[0]))
Observation:
Thought:The customers have been sorted by last name and then first name.
Final Answer: [['Jen', 'Ayai'], ['Harrison', 'Chase'], ['Lang', 'Chain'],
['Elle', 'Elem'], ['Geoff', 'Fusion'], ['Trance', 'Former'], ['Dolly', 'Too']]

> Finished chain.
```

```
"[['Jen', 'Ayai'], ['Harrison', 'Chase'], ['Lang', 'Chain'], ['Elle', 'Elem'],
['Geoff', 'Fusion'], ['Trance', 'Former'], ['Dolly', 'Too']]"
```

```
import langchain
langchain.debug=True
agent.run(f"""Sort these customers by \
last name and then first name \
and print the output: {customer_list}""")
langchain.debug=False
```

```
[chain/start] [1:chain:AgentExecutor] Entering Chain run with input:
{
  "input": "Sort these customers by last name and then first name and print the
output: [['Harrison', 'Chase'], ['Lang', 'Chain'], ['Dolly', 'Too'], ['Elle',
'Elem'], ['Geoff', 'Fusion'], ['Trance', 'Former'], ['Jen', 'Ayai']]"
}
[chain/start] [1:chain:AgentExecutor > 2:chain:LLMChain] Entering Chain run with
input:
{
  "input": "Sort these customers by last name and then first name and print the
output: [['Harrison', 'Chase'], ['Lang', 'Chain'], ['Dolly', 'Too'], ['Elle',
'Elem'], ['Geoff', 'Fusion'], ['Trance', 'Former'], ['Jen', 'Ayai']]",
  "agent_scratchpad": "",
  "stop": [
    "\nObservation:",
    "\n\tObservation:"
  ]
}
[llm/start] [1:chain:AgentExecutor > 2:chain:LLMChain > 3:llm:ChatOpenAI]
Entering LLM run with input:
{
  "prompts": [
    "Human: You are an agent designed to write and execute python code to answer
questions.\nYou have access to a python REPL, which you can use to execute python
code.\nIf you get an error, debug your code and try again.\nOnly use the output
of your code to answer the question. \nYou might know the answer without running
any code, but you should still run the code to get the answer.\nIf it does not
seem like you can write code to answer the question, just return \"I don't know\"
as the answer.\n\n\nPython_REPL: A Python shell. Use this to execute python
commands. Input should be a valid python command. If you want to see the output
of a value, you should print it out with `print(...)`.\n\nUse the following
format:\n\nQuestion: the input question you must answer\nThought: you should
always think about what to do\nAction: the action to take, should be one of
[Python_REPL]\nAction Input: the input to the action\nObservation: the result of
the action\n... (this Thought/Action/Action Input/Observation can repeat N
times)\nThought: I now know the final answer\nFinal Answer: the final answer to
the original input question\n\nBegin!\n\nQuestion: Sort these customers by last
name and then first name and print the output: [['Harrison', 'Chase'], ['Lang',
'Chain'], ['Dolly', 'Too'], ['Elle', 'Elem'], ['Geoff', 'Fusion'], ['Trance',
'Former'], ['Jen', 'Ayai']]\nThought:"
  ]
}
[llm/end] [1:chain:AgentExecutor > 2:chain:LLMChain > 3:llm:ChatOpenAI] [4.59s]
Exiting LLM run with output:
{
  "generations": [
    [
      {
        "text": "I can use the `sorted()` function to sort the list of customers.
I will need to provide a key function that specifies the sorting order based on
last name and then first name.\nAction: Python_REPL\nAction Input:
sorted([['Harrison', 'Chase'], ['Lang', 'Chain'], ['Dolly', 'Too'], ['Elle',
'Elem'], ['Geoff', 'Fusion'], ['Trance', 'Former'], ['Jen', 'Ayai']], key=lambda
x: (x[1], x[0]))",
        "generation_info": {
          "finish_reason": "stop"
        },
        "message": {
          "lc": 1,
          "type": "constructor",
          "id": [
            "langchain",
            "schema",
            "messages",
            "AIMessage"
          ],
          "kwargs": {
            "content": "I can use the `sorted()` function to sort the list of
customers. I will need to provide a key function that specifies the sorting order
based on last name and then first name.\nAction: Python_REPL\nAction Input:
sorted([['Harrison', 'Chase'], ['Lang', 'Chain'], ['Dolly', 'Too'], ['Elle',
'Elem'], ['Geoff', 'Fusion'], ['Trance', 'Former'], ['Jen', 'Ayai']], key=lambda
x: (x[1], x[0]))",
            "additional_kwargs": {}
          }
        }
      }
    ]
  ],
  "llm_output": {
    "token_usage": {
      "prompt_tokens": 328,
      "completion_tokens": 112,
      "total_tokens": 440
    },
    "model_name": "gpt-3.5-turbo"
  },
  "run": null
}
[chain/end] [1:chain:AgentExecutor > 2:chain:LLMChain] [4.59s] Exiting Chain run
with output:
{
  "text": "I can use the `sorted()` function to sort the list of customers. I
will need to provide a key function that specifies the sorting order based on
last name and then first name.\nAction: Python_REPL\nAction Input:
sorted([['Harrison', 'Chase'], ['Lang', 'Chain'], ['Dolly', 'Too'], ['Elle',
'Elem'], ['Geoff', 'Fusion'], ['Trance', 'Former'], ['Jen', 'Ayai']], key=lambda
x: (x[1], x[0]))"
}
[tool/start] [1:chain:AgentExecutor > 4:tool:Python_REPL] Entering Tool run with
input:
"sorted([['Harrison', 'Chase'], ['Lang', 'Chain'], ['Dolly', 'Too'], ['Elle',
'Elem'], ['Geoff', 'Fusion'], ['Trance', 'Former'], ['Jen', 'Ayai']], key=lambda
x: (x[1], x[0]))"
[tool/end] [1:chain:AgentExecutor > 4:tool:Python_REPL] [1.35ms] Exiting Tool run
with output:
""
[chain/start] [1:chain:AgentExecutor > 5:chain:LLMChain] Entering Chain run with
input:
{
  "input": "Sort these customers by last name and then first name and print the
output: [['Harrison', 'Chase'], ['Lang', 'Chain'], ['Dolly', 'Too'], ['Elle',
'Elem'], ['Geoff', 'Fusion'], ['Trance', 'Former'], ['Jen', 'Ayai']]",
  "agent_scratchpad": "I can use the `sorted()` function to sort the list of
customers. I will need to provide a key function that specifies the sorting order
based on last name and then first name.\nAction: Python_REPL\nAction Input:
sorted([['Harrison', 'Chase'], ['Lang', 'Chain'], ['Dolly', 'Too'], ['Elle',
'Elem'], ['Geoff', 'Fusion'], ['Trance', 'Former'], ['Jen', 'Ayai']], key=lambda
x: (x[1], x[0]))\nObservation: \nThought:",
  "stop": [
    "\nObservation:",
    "\n\tObservation:"
  ]
```

```
}
[llm/start] [1:chain:AgentExecutor > 5:chain:LLMChain > 6:llm:ChatOpenAI]
Entering LLM run with input:
{
  "prompts": [
    "Human: You are an agent designed to write and execute python code to answer
questions.\nYou have access to a python REPL, which you can use to execute python
code.\nIf you get an error, debug your code and try again.\nOnly use the output
of your code to answer the question. \nYou might know the answer without running
any code, but you should still run the code to get the answer.\nIf it does not
seem like you can write code to answer the question, just return \"I don't know\"
as the answer.\n\n\nPython_REPL: A Python shell. Use this to execute python
commands. Input should be a valid python command. If you want to see the output
of a value, you should print it out with `print(...)`.\n\nUse the following
format:\n\nQuestion: the input question you must answer\nThought: you should
always think about what to do\nAction: the action to take, should be one of
[Python_REPL]\nAction Input: the input to the action\nObservation: the result of
the action\n... (this Thought/Action/Action Input/Observation can repeat N
times)\nThought: I now know the final answer\nFinal Answer: the final answer to
the original input question\n\nBegin!\n\nQuestion: Sort these customers by last
name and then first name and print the output: [['Harrison', 'Chase'], ['Lang',
'Chain'], ['Dolly', 'Too'], ['Elle', 'Elem'], ['Geoff', 'Fusion'], ['Trance',
'Former'], ['Jen', 'Ayai']]\nThought:I can use the `sorted()` function to sort
the list of customers. I will need to provide a key function that specifies the
sorting order based on last name and then first name.\nAction:
Python_REPL\nAction Input: sorted([['Harrison', 'Chase'], ['Lang', 'Chain'],
['Dolly', 'Too'], ['Elle', 'Elem'], ['Geoff', 'Fusion'], ['Trance', 'Former'],
['Jen', 'Ayai']], key=lambda x: (x[1], x[0]))\nObservation: \nThought:"
  ]
}
[llm/end] [1:chain:AgentExecutor > 5:chain:LLMChain > 6:llm:ChatOpenAI] [3.89s]
Exiting LLM run with output:
{
  "generations": [
    [
      {
        "text": "The customers have been sorted by last name and then first
name.\nFinal Answer: [['Jen', 'Ayai'], ['Harrison', 'Chase'], ['Lang', 'Chain'],
['Elle', 'Elem'], ['Geoff', 'Fusion'], ['Trance', 'Former'], ['Dolly', 'Too']]",
        "generation_info": {
          "finish_reason": "stop"
        },
        "message": {
          "lc": 1,
          "type": "constructor",
          "id": [
            "langchain",
            "schema",
            "messages",
            "AIMessage"
          ],
          "kwargs": {
            "content": "The customers have been sorted by last name and then
first name.\nFinal Answer: [['Jen', 'Ayai'], ['Harrison', 'Chase'], ['Lang',
'Chain'], ['Elle', 'Elem'], ['Geoff', 'Fusion'], ['Trance', 'Former'], ['Dolly',
'Too']]",
            "additional_kwargs": {}
          }
        }
      }
    ]
  ],
  "llm_output": {
    "token_usage": {
      "prompt_tokens": 445,
      "completion_tokens": 67,
      "total_tokens": 512
    },
    "model_name": "gpt-3.5-turbo"
  },
  "run": null
}
[chain/end] [1:chain:AgentExecutor > 5:chain:LLMChain] [3.89s] Exiting Chain run
with output:
{
  "text": "The customers have been sorted by last name and then first
name.\nFinal Answer: [['Jen', 'Ayai'], ['Harrison', 'Chase'], ['Lang', 'Chain'],
['Elle', 'Elem'], ['Geoff', 'Fusion'], ['Trance', 'Former'], ['Dolly', 'Too']]"
}
[chain/end] [1:chain:AgentExecutor] [8.49s] Exiting Chain run with output:
{
  "output": "[['Jen', 'Ayai'], ['Harrison', 'Chase'], ['Lang', 'Chain'], ['Elle',
'Elem'], ['Geoff', 'Fusion'], ['Trance', 'Former'], ['Dolly', 'Too']]"
}
```

3. 定义自己的工具并在代理中使用

```python
# 导入tool函数装饰器
from langchain.agents import tool
from datetime import date

@tool
def time(text: str) -> str:
    """Returns todays date, use this for any \
    questions related to knowing todays date. \
    The input should always be an empty string, \
    and this function will always return todays \
    date - any date mathmatics should occur \
    outside this function."""
    return str(date.today())

agent= initialize_agent(
    tools + [time], #将刚刚创建的时间工具加入到已有的工具中
    llm, #初始化的模型
    agent=AgentType.CHAT_ZERO_SHOT_REACT_DESCRIPTION,  #代理类型
    handle_parsing_errors=True, #处理解析错误
    verbose = True #输出中间步骤
)

agent("whats the date today?")
```

~~~
> Entering new AgentExecutor chain...
Question: What's the date today?
Thought: I can use the `time` tool to get the current date.
Action:
```
{
  "action": "time",
  "action_input": ""
}
```
Observation: 2023-08-09
Thought:I now know the final answer.
Final Answer: The date today is 2023-08-09.

> Finished chain.
~~~

```
{'input': 'whats the date today?', 'output': 'The date today is 2023-08-09.'}
```