五、使用LangGraph构建多智能体工作流

--楼兰

上一章节我们在没有大模型的加持下，全面演练了LangGraph的Graph图结构。 这一章节，就结合大模型，来深入理解LangGraph如何通过Graph构建复杂的大模型应用。

一、流式输出大模型调用结果

在介绍Graph的流式输出时，我们提到LangGraph的Graph流式输出有几种不同的模式，其中有一种messages模式，是用来监控大语言模型的Token记录的。这里我们就可以来测试一下。

```
from config.load_key import load_key
from langchain_community.chat_models import ChatTongyi
# 构建阿里云百炼大模型客户端
llm = ChatTongyi(
    model="qwen-plus",
    api_key=load_key("BAILIAN_API_KEY"),
)

from langgraph.graph import StateGraph, MessagesState, START

from langgraph.checkpoint.memory import InMemorySaver

def call_model(state: MessagesState):
    response = llm.invoke(state["messages"])
    return {"messages": response}


builder = StateGraph(MessagesState)
builder.add_node(call_model)
builder.add_edge(START, "call_model")


graph = builder.compile()


for chunk in graph.stream(
    {"messages": [{"role": "user", "content": "湖南的省会是哪里？"}]},
    stream_mode="messages",
):
    print(chunk)
```

```
(AIMessageChunk(content='湖南省', additional_kwargs={}, response_metadata={}, id='run-0801c41e-9a01-46a5-b325-3bfbaf998031'), {'langgraph_step': 1, 'langgraph_node': 'call_model', 'langgraph_triggers': ('branch:to:call_model',), 'langgraph_path': ('__pregel_pull', 'call_model'), 'langgraph_checkpoint_ns': 'call_model:184f6a2f-f413-ccf3-8de0-9e29d823fd29', 'checkpoint_ns': 'call_model:184f6a2f-f413-ccf3-8de0-9e29d823fd29', 'ls_provider': 'tongyi', 'ls_model_type': 'chat', 'ls_model_name': 'qwen-plus'})
(AIMessageChunk(content='的', additional_kwargs={}, response_metadata={}, id='run-0801c41e-9a01-46a5-b325-3bfbaf998031'), {'langgraph_step': 1, 'langgraph_node': 'call_model', 'langgraph_triggers': ('branch:to:call_model',), 'langgraph_path': ('__pregel_pull', 'call_model'), 'langgraph_checkpoint_ns': 'call_model:184f6a2f-f413-ccf3-8de0-9e29d823fd29', 'checkpoint_ns': 'call_model:184f6a2f-f413-ccf3-8de0-9e29d823fd29', 'ls_provider': 'tongyi', 'ls_model_type': 'chat', 'ls_model_name': 'qwen-plus'})
(AIMessageChunk(content='省', additional_kwargs={}, response_metadata={}, id='run-0801c41e-9a01-46a5-b325-3bfbaf998031'), {'langgraph_step': 1, 'langgraph_node': 'call_model', 'langgraph_triggers': ('branch:to:call_model',), 'langgraph_path': ('__pregel_pull', 'call_model'), 'langgraph_checkpoint_ns': 'call_model:184f6a2f-f413-ccf3-8de0-9e29d823fd29', 'checkpoint_ns': 'call_model:184f6a2f-f413-ccf3-8de0-9e29d823fd29', 'ls_provider': 'tongyi', 'ls_model_type': 'chat', 'ls_model_name': 'qwen-plus'})
(AIMessageChunk(content='会是长沙市。', additional_kwargs={}, response_metadata={}, id='run-0801c41e-9a01-46a5-b325-3bfbaf998031'), {'langgraph_step': 1, 'langgraph_node': 'call_model', 'langgraph_triggers': ('branch:to:call_model',), 'langgraph_path': ('__pregel_pull', 'call_model'), 'langgraph_checkpoint_ns': 'call_model:184f6a2f-f413-ccf3-8de0-9e29d823fd29', 'checkpoint_ns': 'call_model:184f6a2f-f413-ccf3-8de0-9e29d823fd29', 'ls_provider': 'tongyi', 'ls_model_type': 'chat', 'ls_model_name': 'qwen-plus'})
(AIMessageChunk(content='长沙是中国历史文化名', additional_kwargs={}, response_metadata={}, id='run-0801c41e-9a01-46a5-b325-3bfbaf998031'), {'langgraph_step': 1, 'langgraph_node': 'call_model', 'langgraph_triggers': ('branch:to:call_model',), 'langgraph_path': ('__pregel_pull', 'call_model'), 'langgraph_checkpoint_ns': 'call_model:184f6a2f-f413-ccf3-8de0-9e29d823fd29', 'checkpoint_ns': 'call_model:184f6a2f-f413-ccf3-8de0-9e29d823fd29', 'ls_provider': 'tongyi', 'ls_model_type': 'chat', 'ls_model_name': 'qwen-plus'})
(AIMessageChunk(content='城，也是一座', additional_kwargs={}, response_metadata={}, id='run-0801c41e-9a01-46a5-b325-3bfbaf998031'), {'langgraph_step': 1, 'langgraph_node': 'call_model', 'langgraph_triggers': ('branch:to:call_model',), 'langgraph_path': ('__pregel_pull', 'call_model'), 'langgraph_checkpoint_ns': 'call_model:184f6a2f-f413-ccf3-8de0-9e29d823fd29', 'checkpoint_ns': 'call_model:184f6a2f-f413-ccf3-8de0-9e29d823fd29', 'ls_provider': 'tongyi', 'ls_model_type': 'chat', 'ls_model_name': 'qwen-plus'})
(AIMessageChunk(content='充满活力的现代化', additional_kwargs={}, response_metadata={}, id='run-0801c41e-9a01-46a5-b325-3bfbaf998031'), {'langgraph_step': 1, 'langgraph_node': 'call_model', 'langgraph_triggers': ('branch:to:call_model',), 'langgraph_path': ('__pregel_pull', 'call_model'), 'langgraph_checkpoint_ns': 'call_model:184f6a2f-f413-ccf3-8de0-9e29d823fd29', 'checkpoint_ns': 'call_model:184f6a2f-f413-ccf3-8de0-9e29d823fd29', 'ls_provider': 'tongyi', 'ls_model_type': 'chat', 'ls_model_name': 'qwen-plus'})
```

```
(AIMessageChunk(content='城市，位于湖南', additional_kwargs={}, response_metadata={},
id='run-0801c41e-9a01-46a5-b325-3bfbaf998031'), {'langgraph_step': 1, 'langgraph_node':
'call_model', 'langgraph_triggers': ('branch:to:call_model',), 'langgraph_path':
('__pregel_pull', 'call_model'), 'langgraph_checkpoint_ns': 'call_model:184f6a2f-f413-
ccf3-8de0-9e29d823fd29', 'checkpoint_ns': 'call_model:184f6a2f-f413-ccf3-8de0-
9e29d823fd29', 'ls_provider': 'tongyi', 'ls_model_type': 'chat', 'ls_model_name': 'qwen-
plus'})
(AIMessageChunk(content='东部偏北、', additional_kwargs={}, response_metadata={}, id='run-
0801c41e-9a01-46a5-b325-3bfbaf998031'), {'langgraph_step': 1, 'langgraph_node':
'call_model', 'langgraph_triggers': ('branch:to:call_model',), 'langgraph_path':
('__pregel_pull', 'call_model'), 'langgraph_checkpoint_ns': 'call_model:184f6a2f-f413-
ccf3-8de0-9e29d823fd29', 'checkpoint_ns': 'call_model:184f6a2f-f413-ccf3-8de0-
9e29d823fd29', 'ls_provider': 'tongyi', 'ls_model_type': 'chat', 'ls_model_name': 'qwen-
plus'})
(AIMessageChunk(content='湘江下游和', additional_kwargs={}, response_metadata={}, id='run-
0801c41e-9a01-46a5-b325-3bfbaf998031'), {'langgraph_step': 1, 'langgraph_node':
'call_model', 'langgraph_triggers': ('branch:to:call_model',), 'langgraph_path':
('__pregel_pull', 'call_model'), 'langgraph_checkpoint_ns': 'call_model:184f6a2f-f413-
ccf3-8de0-9e29d823fd29', 'checkpoint_ns': 'call_model:184f6a2f-f413-ccf3-8de0-
9e29d823fd29', 'ls_provider': 'tongyi', 'ls_model_type': 'chat', 'ls_model_name': 'qwen-
plus'})
(AIMessageChunk(content='洞庭湖以', additional_kwargs={}, response_metadata={}, id='run-
0801c41e-9a01-46a5-b325-3bfbaf998031'), {'langgraph_step': 1, 'langgraph_node':
'call_model', 'langgraph_triggers': ('branch:to:call_model',), 'langgraph_path':
('__pregel_pull', 'call_model'), 'langgraph_checkpoint_ns': 'call_model:184f6a2f-f413-
ccf3-8de0-9e29d823fd29', 'checkpoint_ns': 'call_model:184f6a2f-f413-ccf3-8de0-
9e29d823fd29', 'ls_provider': 'tongyi', 'ls_model_type': 'chat', 'ls_model_name': 'qwen-
plus'})
(AIMessageChunk(content='南地区。', additional_kwargs={}, response_metadata={}, id='run-
0801c41e-9a01-46a5-b325-3bfbaf998031'), {'langgraph_step': 1, 'langgraph_node':
'call_model', 'langgraph_triggers': ('branch:to:call_model',), 'langgraph_path':
('__pregel_pull', 'call_model'), 'langgraph_checkpoint_ns': 'call_model:184f6a2f-f413-
ccf3-8de0-9e29d823fd29', 'checkpoint_ns': 'call_model:184f6a2f-f413-ccf3-8de0-
9e29d823fd29', 'ls_provider': 'tongyi', 'ls_model_type': 'chat', 'ls_model_name': 'qwen-
plus'})
(AIMessageChunk(content='', additional_kwargs={}, response_metadata={'finish_reason':
'stop', 'request_id': '8934e9a2-f4f5-9d0c-982e-ef689e335e7c', 'token_usage':
{'input_tokens': 15, 'output_tokens': 38, 'total_tokens': 53, 'prompt_tokens_details':
{'cached_tokens': 0}}}, id='run-0801c41e-9a01-46a5-b325-3bfbaf998031'), {'langgraph_step':
1, 'langgraph_node': 'call_model', 'langgraph_triggers': ('branch:to:call_model',),
'langgraph_path': ('__pregel_pull', 'call_model'), 'langgraph_checkpoint_ns':
'call_model:184f6a2f-f413-ccf3-8de0-9e29d823fd29', 'checkpoint_ns': 'call_model:184f6a2f-
f413-ccf3-8de0-9e29d823fd29', 'ls_provider': 'tongyi', 'ls_model_type': 'chat',
'ls_model_name': 'qwen-plus'})
```

通常，如果要对大模型调用成本进行统计时，这种messages就是比较好的一种方式。

二、大模型消息持久化

和之前介绍LangGraph的Agent相似，Graph图也支持构建消息的持久化功能。并且也通常支持通过checkpointer构建短期记忆，以store构建长期记忆。
这里短期记忆和长期记忆，都是可以通过内存或者数据库进行持久化保存的。不过短期记忆更倾向于通过对消息的短期存储，实现多轮对话的效果。而长期记忆则倾向于对消息长期存储后支持语义检索。

```
from config.load_key import load_key
from langchain_community.chat_models import ChatTongyi
# 构建阿里云百炼大模型客户端
llm = ChatTongyi(
    model="qwen-plus",
    api_key=load_key("BAILIAN_API_KEY"),
)

from langgraph.graph import StateGraph, MessagesState, START

from langgraph.checkpoint.memory import InMemorySaver

def call_model(state: MessagesState):
    response = llm.invoke(state["messages"])
    return {"messages": response}


builder = StateGraph(MessagesState)
builder.add_node(call_model)
builder.add_edge(START, "call_model")

checkpointer = InMemorySaver()
graph = builder.compile(checkpointer=checkpointer)

config = {
    "configurable": {
        "thread_id": "1"
    }
}

for chunk in graph.stream(
    {"messages": [{"role": "user", "content": "湖南的省会是哪里？"}]},
    config,
    stream_mode="values",
):
    chunk["messages"][-1].pretty_print()

for chunk in graph.stream(
    {"messages": [{"role": "user", "content": "湖北呢？"}]},
    config,
    stream_mode="values",
):
    chunk["messages"][-1].pretty_print()
```

```
================================•[1m Human Message •[0m================================

湖南的省会是哪里?
==================================•[1m Ai Message •[0m==================================

湖南省的省会是长沙市。长沙是中国中部地区的重要城市，也是湖南省的政治、经济、文化和交通中心。
================================•[1m Human Message •[0m================================

湖北呢?
==================================•[1m Ai Message •[0m==================================

湖北省的省会是武汉市。武汉是中国中部地区的重要城市，也是湖北省的政治、经济、文化和交通中心。武汉地处长江中游，素有“九省通衢”之称，是中国内陆最大的水陆空交通枢纽。
```

LangGraph中围绕Checkpoint短期记忆，提供了非常丰富的补充功能。

三、Human-In-Loop人类干预

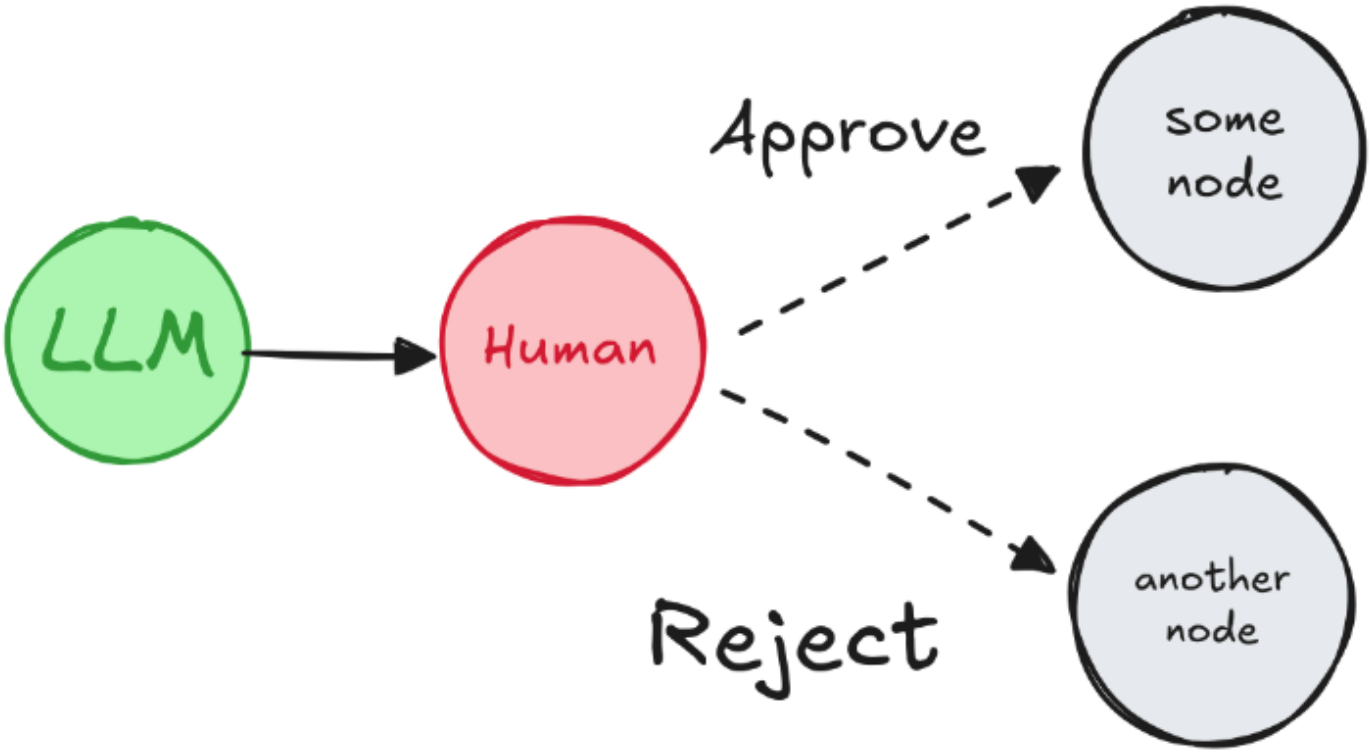
在LangGraph中也可以通过中断任务，等待确认的方式，来实现过程干预，这样能够更好的减少大语言模型的结果不稳定给任务带来的影响。

在具体实现人类干预时，需要注意一下几点：

- 必须指定一个checkpointer短期记忆，否则无法保存任务状态。
- 在执行Graph任务时，必须指定一个带有thread_id的配置项，指定线程ID。之后才能通过线程ID，指定恢复线程。
- 在任务执行过程中，通过interrupt()方法，中断任务，等待确认。
- 在人类确认之后，使用Graph提交一个resume=True的Command指令，恢复任务，并继续进行。

这种实现方式，在之前介绍LangGraph构建单Agent时已经介绍过，不过，结合Graph的State，在多个Node之间进行复杂控制，这样更能体现出人类监督的价值。

例如，下面的案例可以实现这样一种典型的人类确认：




```
# 构建一个带有Human-In-Loop的图
from operator import add

from langchain_core.messages import AnyMessage
from langgraph.checkpoint.memory import InMemorySaver
from langgraph.constants import START, END
from langgraph.graph import StateGraph

from config.load_key import load_key
from langchain_community.chat_models import ChatTongyi
# 构建阿里云百炼大模型客户端
llm = ChatTongyi(
    model="qwen-plus",
    api_key=load_key("BAILIAN_API_KEY"),
)

from typing import Literal, TypedDict, Annotated
from langgraph.types import interrupt, Command

class State(TypedDict):
    messages: Annotated[list[AnyMessage], add]
def human_approval(state: State) -> Command[Literal["call_llm", END]]:
    is_approved = interrupt(
        {
            "question": "是否同意调用大语言模型？"
        }
    )

    if is_approved:
```

```
        return Command(goto="call_llm")
    else:
        return Command(goto=END)

def call_llm(state:State):
    response = llm.invoke(state["messages"])
    return {"messages": [response]}

builder = StateGraph(State)

# Add the node to the graph in an appropriate location
# and connect it to the relevant nodes.
builder.add_node("human_approval", human_approval)
builder.add_node("call_llm",call_llm)

builder.add_edge(START,"human_approval")

checkpointer = InMemorySaver()

graph = builder.compile(checkpointer=checkpointer)
```

```
from langchain_core.messages import HumanMessage

# 提交任务，等待确认
thread_config = {"configurable": {"thread_id": 1}}
graph.invoke({"messages": [HumanMessage("湖南的省会是哪里？")]}, config=thread_config)
# 执行后会中断任务，等待确认
```

```
{'messages': [HumanMessage(content='湖南的省会是哪里？', additional_kwargs={},
response_metadata={}),
  AIMessage(content='湖南省的省会是**长沙市**。长沙是中国中部地区的重要城市，也是湖南的政治、经济、文化和交通中心。长沙历史悠久，文化底蕴深厚，著名景点包括岳麓山、橘子洲头、马王堆汉墓等。同时，长沙也以美食闻名，如臭豆腐、剁椒鱼头等。', additional_kwargs={}, response_metadata={'model_name': 'qwen-plus',
'finish_reason': 'stop', 'request_id': '1421ad01-bb72-9267-82b1-c2790e8ca292',
'token_usage': {'input_tokens': 15, 'output_tokens': 69, 'total_tokens': 84,
'prompt_tokens_details': {'cached_tokens': 0}}}, id='run-61a39cd8-55cb-46a1-9662-
7712aa059e56-0'),
  HumanMessage(content='湖南的省会是哪里？', additional_kwargs={}, response_metadata={})],
 '__interrupt__': [Interrupt(value={'question': '是否同意调用大语言模型？'}, resumable=True,
ns=['human_approval:b1a44bb4-a3e3-04ab-a5c3-9454dd35d9ca'])]}
```

```
# 确认同意，继续执行任务
# final_result = graph.invoke(Command(resume=True), config=thread_config)
# print(final_result)
# 不同意，终止任务
final_result = graph.invoke(Command(resume=False), config=thread_config)
print(final_result)
```

```
{'messages': [HumanMessage(content='湖南的省会是哪里？', additional_kwargs={},
response_metadata={}), AIMessage(content='湖南省的省会是**长沙市**。长沙是中国中部地区的重要城市，
也是湖南的政治、经济、文化和交通中心。长沙历史悠久，文化底蕴深厚，著名景点包括岳麓山、橘子洲头、马王堆汉墓
等。同时，长沙也以美食闻名，如臭豆腐、剁椒鱼头等。', additional_kwargs={}, response_metadata=
{'model_name': 'qwen-plus', 'finish_reason': 'stop', 'request_id': '1421ad01-bb72-9267-
82b1-c2790e8ca292', 'token_usage': {'input_tokens': 15, 'output_tokens': 69,
'total_tokens': 84, 'prompt_tokens_details': {'cached_tokens': 0}}}, id='run-61a39cd8-
55cb-46a1-9662-7712aa059e56-0'), HumanMessage(content='湖南的省会是哪里？',
additional_kwargs={}, response_metadata={})]}
```

注意：

- 任务中断和恢复，需要保持相同的thread_id。通常应用当中都会单独生成一个随机的thread_id，保证唯一的同时，防止其他任务干扰。
- interrupt()方法中断任务的时间不能过长，过长了之后就无法恢复任务了。
- 任务确认时，Command中传递的resume可以是简单的True或False，也可以是一个字典。通过字典可以进行更多的判断。

四、Time Travel时间回溯

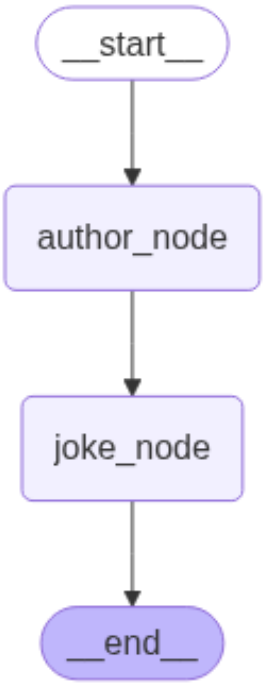
由于大语言模型回答问题的不确定性，基于大语言模型构建的应用，也是充满不确定性的。而对于这种不确定性的系统，就有必要进行更精确的检查。当某一个步骤出现问题时，才能及时发现问题，并从发现问题的那个步骤进行重演。为此，LangGraph提供了Time Travel时间回溯功能，可以保存Graph的运行过程，并可以手动指定从Graph的某一个Node开始进行重演。

具体实现时，需要注意以下几点：

- 在运行Graph时，需要提供初始的输入消息。
- 运行时，指定thread_id线程ID。并且要基于这个线程ID，再指定一个checkpoint检查点。执行后将在每一个Node执行后，生成一个check_point_id
- 指定thread_id和check_point_id，进行任务重演。重演前，可以选择更新state，当然，如果没问题，也可以不指定。

```
# 构建一个图。图中两个步骤：第一步让大模型推荐一个有名的作家，第二步，让大模型用推荐的作家的风格写一个100字以内的笑话。
from typing import TypedDict
from typing_extensions import NotRequired
from langgraph.checkpoint.memory import InMemorySaver
from langgraph.constants import START, END
from langgraph.graph import StateGraph
```

```python
from config.load_key import load_key
from langchain_community.chat_models import ChatTongyi
# 构建阿里云百炼大模型客户端
llm = ChatTongyi(
    model="qwen-plus",
    api_key=load_key("BAILIAN_API_KEY"),
)

class State(TypedDict):
    author: NotRequired[str]
    joke: NotRequired[str]

def author_node(state:State):
    prompt = "帮我推荐一位受人们欢迎的作家。只需要给出作家的名字即可。"
    author = llm.invoke(prompt)
    return {"author":author}

def joke_node(state:State):
    prompt = f"用作家：{state['author']} 的风格，写一个100字以内的笑话"
    joke = llm.invoke(prompt)
    return {"joke":joke}

builder = StateGraph(State)
builder.add_node(author_node)
builder.add_node(joke_node)

builder.add_edge(START,"author_node")
builder.add_edge("author_node","joke_node")
builder.add_edge("joke_node",END)

checkpointer = InMemorySaver()
graph = builder.compile(checkpointer=checkpointer)
graph
```




```
# 正常执行一个图
import uuid

config = {
    "configurable": {
        "thread_id": uuid.uuid4(),
    }
}
state = graph.invoke({}, config)

print(state["author"])
print()
print(state["joke"])
```

```
content='村上春树' additional_kwargs={} response_metadata={'model_name': 'qwen-plus',
'finish_reason': 'stop', 'request_id': '2b3b9e85-a2d1-9ecb-b1c7-85cfa7716009',
'token_usage': {'input_tokens': 23, 'output_tokens': 4, 'total_tokens': 27,
'prompt_tokens_details': {'cached_tokens': 0}}} id='run-a4d36728-d5f7-4e82-8ff7-
b78a9463ee89-0'

content='一只羊走进了酒吧，对 bartender 说：“来杯啤酒。”  \nBartender 奇怪地问：“你怎么一个人来了？”
\n羊叹了口气，说：“因为我太太在看村上春树的小说，她说今晚不想被打扰……”  \n它喝了一口啤酒，望着窗外，低声自
语：“其实我也不懂那些莫名其妙的猫和森林，但至少她喜欢安静的世界。”' additional_kwargs={}
response_metadata={'model_name': 'qwen-plus', 'finish_reason': 'stop', 'request_id':
'58c36f68-2439-989a-88c4-d2e858a9a665', 'token_usage': {'input_tokens': 166,
'output_tokens': 85, 'total_tokens': 251, 'prompt_tokens_details': {'cached_tokens': 0}}}
id='run-fd1a1d69-16fe-4efe-8f57-b3cd5a5a3165-0'
```

```python
# 查看所有checkpoint检查点
states = list(graph.get_state_history(config))

for state in states:
    print(state.next)
    print(state.config["configurable"]["checkpoint_id"])
    print()
```

```
()
1f048581-3a14-6244-8002-d1d24e3f61b9

('joke_node',)
1f048581-142d-6184-8001-cd9d83040202

('author_node',)
1f048581-0ed3-6850-8000-1ed07178fe37

('__start__',)
1f048581-0ed0-618c-bfff-b4880376d7cc
```

```python
# 选定某一个检查点。这里选择author_node，让大模型重新推荐作家
selected_state = states[1]
print(selected_state.next)
print(selected_state.values)
```

```
('joke_node',)
{'author': AIMessage(content='村上春树', additional_kwargs={}, response_metadata=
{'model_name': 'qwen-plus', 'finish_reason': 'stop', 'request_id': '2b3b9e85-a2d1-9ecb-
b1c7-85cfa7716009', 'token_usage': {'input_tokens': 23, 'output_tokens': 4,
'total_tokens': 27, 'prompt_tokens_details': {'cached_tokens': 0}}}, id='run-a4d36728-
d5f7-4e82-8ff7-b78a9463ee89-0')}
```

```python
# 为了后面的重演，更新state。可选步骤：
new_config = graph.update_state(selected_state.config, values={"author": "郭德纲"})
print(new_config)
```

```
{'configurable': {'thread_id': '97e870bc-2243-4a60-937a-b51f4c446873', 'checkpoint_ns':
'', 'checkpoint_id': '1f048586-7054-6d18-8002-780840baa24b'}}
```

```
# 接下来，指定thread_id和checkpoint_id，进行重演
graph.invoke(None,new_config)
```

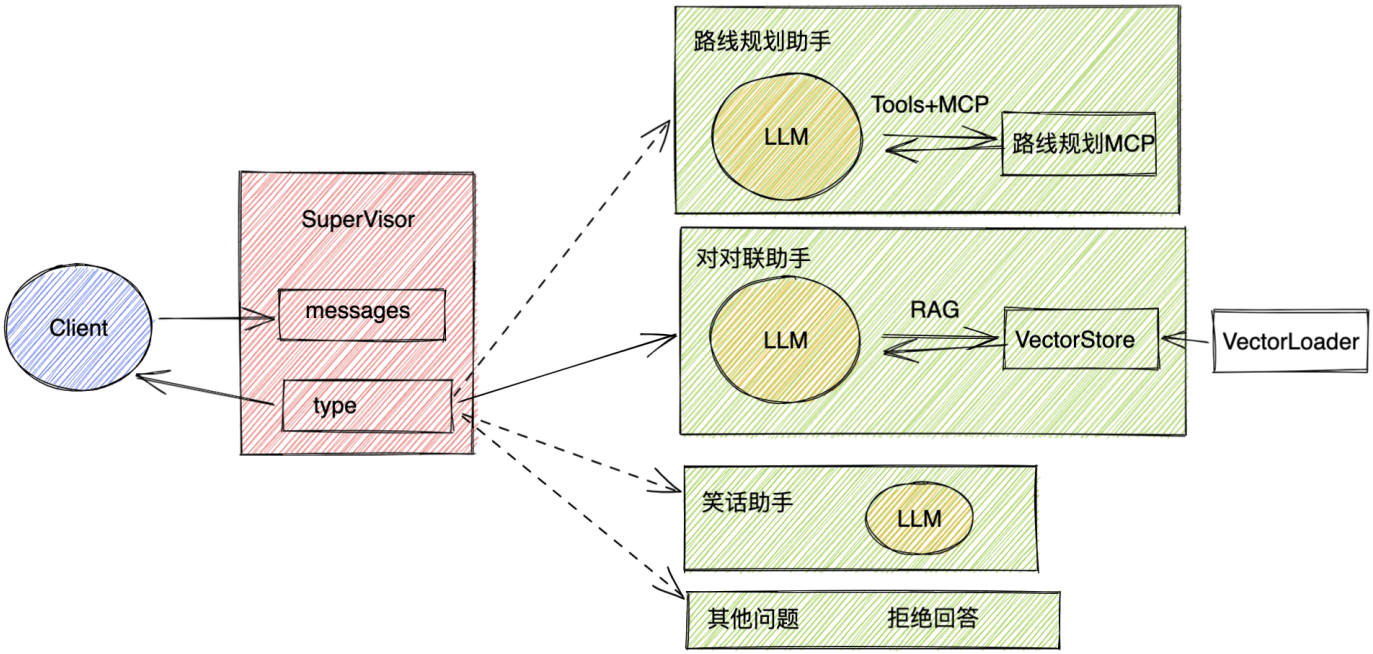
```
{'author': '郭德纲',
 'joke': AIMessage(content='话说这有个主儿，非得让郭老师给写个笑话。您瞧瞧，这就来了——\n\n有位爷去买西瓜，问：“这瓜甜不？” 卖瓜的说：“甜！不甜你砸我！” 结果爷买了瓜，切开一尝，苦的。他拎着瓜回去，二话不说，直接把瓜往卖瓜的脑袋上一扣：“不是说让我砸你吗？我这不是成全你呢嘛！” \n\n得嘞，各位，乐呵乐呵，别生气啊！', additional_kwargs={}, response_metadata={'model_name': 'qwen-plus', 'finish_reason': 'stop', 'request_id': 'f59ec19d-1902-9baf-8f3d-fe1c9b78ef93', 'token_usage': {'input_tokens': 26, 'output_tokens': 117, 'total_tokens': 143, 'prompt_tokens_details': {'cached_tokens': 0}}}, id='run-a8692f61-41d9-4db0-bf49-a7cd84beb336-0')}
```

五、多智能体架构

可以看到，在LangChain体系中，LangChain主要集成了和大语言模型交互的能力，而LangGraph主要实现了复杂的流程调度。将这两个能力结合起来，一个强大的多智能体构建框架就已经成型了。

接下来，我们就用LangGraph来实现一个非常典型的多智能体架构，作为一个完整的案例。

- 这个机器人可以通过一个supervisor节点，对用户的输入进行分类，然后根据分类结果，选择不同的agent节点进行处理。
- 接下来每个Agent节点，都可以选择不同的工具进行处理，最后将处理结果汇总，返回给supervisor节点。
- supervisor节点再将结果返回给用户。



在实现时，为了能够更综合的演练这么长时间的学习效果，我们在对各个智能体的功能进行了一些设计，从而让这个小案例不再只是一个简单的Demo。

- 其他问题，只添加一个简单的响应结果。
- 笑话助手，直接与大模型交互获得一个结果。
- 对对联助手，从向量数据库中获取补充的资料，实现一个典型的RAG流程。
- 路线规划助手，则需要调度外部的MCP服务，获取补充信息。

这个案例，即作为LangGraph系列的总结演练，也作为一个典型的多智能体案例，强烈建议你，自己动手试试实现一个。在这个案例中，LangGraph更多的帮助我们来梳理各个智能体之间如何协调。而具体实现时，可以更多的借鉴LangChain的能力。还有，不要忘了，LangGraph还提供了很多开发过程中可以用到的工具，比如自定义流式输出、Time-travel时间重演等，都可以在这个案例中逐步尝试。

> 最终代码参见视频

总结

从LangGraph的整个演练过程可以看到， LangGraph的核心是Graph。Graph其实是一个与大模型没有直接关联的，处理复杂任务的流程结构。LangGraph或者说整个LangChain系列，其实是将传统的软件构建经验与大语言模型的能力进行结合，从而进一步打造出强大的智能体，解决更多实际的复杂问题。这也进一步验证了，大语言模型未来的发展方向，一定是需要与传统应用相结合，这样才能更好的发挥大语言模型的价值。而这，或许是LangChain系列最核心的价值所在。