

Agent, Assistant và Message

Trong **LangChain**, các loại **Message** đóng vai trò rất quan trọng khi làm việc với các mô hình như ChatOpenAI, nhất là khi xây dựng các hệ thống hội thoại (conversational agents). Những message này mô phỏng cách người và AI giao tiếp với nhau trong một cuộc trò chuyện.

- [Giới thiệu](#)
- [Assistant và Agent](#)
- [HumanMessage - Tin nhắn từ người dùng](#)
- [AIMessage - Tin nhắn phản hồi từ AI](#)
- [SystemMessage - Thiết lập ngữ cảnh](#)
- [ToolMessage - Kết quả phản hồi từ một công cụ \(tool\)](#)
- [ToolCall - Mô hình ngôn ngữ gọi đến một công cụ](#)

Giới thiệu

Trong LangChain, **Message** là thành phần cốt lõi dùng để mô tả các tương tác giữa người dùng, hệ thống và mô hình ngôn ngữ (LLM). Việc hiểu rõ các loại Message giúp bạn xây dựng các tác vụ hội thoại, agent, và ứng dụng AI một cách chính xác, dễ kiểm soát và có thể mở rộng.

LangChain định nghĩa nhiều loại message khác nhau, mỗi loại phản ánh một vai trò cụ thể trong quá trình giao tiếp. Dưới đây là tổng quan các loại message chính:

1. HumanMessage – Tin nhắn từ người dùng

Đây là loại message đại diện cho đầu vào từ người dùng cuối. Khi bạn nhập văn bản để hỏi AI, LangChain sẽ đóng gói nội dung đó thành một `HumanMessage`.

```
from langchain.schema import HumanMessage

message = HumanMessage(content="Bạn có thể giúp tôi tính 5 nhân 3 không?")
```

Dùng khi mô phỏng hoặc truyền input từ con người.

2. AIMessage – Phản hồi từ AI

Đại diện cho phản hồi từ LLM, thường là kết quả sau khi xử lý `HumanMessage`. Ngoài văn bản, `AIMessage` còn có thể bao gồm **tool calls**, nếu AI quyết định gọi một công cụ thay vì trả lời trực tiếp.

```
from langchain.schema import AIMessage

message = AIMessage(content="Kết quả là 15.")
```

Dùng để mô tả output từ AI.

3. SystemMessage – Định hướng AI

Được dùng để thiết lập bối cảnh hoặc hướng dẫn hành vi của AI từ đầu. Những chỉ dẫn như “bạn là một trợ lý thân thiện” hay “chỉ trả lời bằng tiếng Việt” thường được đặt trong `SystemMessage`.

```
from langchain.schema import SystemMessage
```

```
message = SystemMessage(content="Bạn là một trợ lý AI chuyên nghiệp.")
```

Dùng để hướng dẫn mô hình trước khi bắt đầu cuộc trò chuyện.

4. ToolMessage – Phản hồi từ công cụ (tool)

Khi một AI gọi tool (hàm toán học, API, công cụ bên ngoài...), kết quả trả về sẽ được biểu diễn qua ToolMessage. Nó mô tả output của tool, và giúp AI tiếp tục hội thoại dựa trên dữ liệu đó.

```
from langchain.schema import ToolMessage
```

```
message = ToolMessage(  
    tool_call_id="abc123",  
    content="Kết quả của phép chia là 2.5"  
)
```

Dùng khi bạn có workflow với tool calling.

5. ToolCall – Yêu cầu từ AI gọi một tool

Không phải là message, nhưng thường được **gói trong AIMessage**. Khi AI muốn gọi một hàm cụ thể, nó tạo ra một ToolCall để mô tả tên tool và các tham số truyền vào.

```
from langchain.schema.messages import ToolCall
```

```
call = ToolCall(  
    name="multiply",  
    args={"a": 5, "b": 3},  
    id="tool_call_1"  
)
```

Gắn liền với quá trình sử dụng `bind_tools()` hoặc khi xây dựng các agent có tool.

Tóm lại

Message Type	Vai trò chính	Dùng khi nào?
HumanMessage	Người dùng gửi yêu cầu	Mọi input của người dùng

Message Type	Vai trò chính	Dùng khi nào?
AIMessage	AI phản hồi hoặc gọi tool	Output từ AI
SystemMessage	Thiết lập bối cảnh và hành vi AI	Trước khi bắt đầu cuộc trò chuyện
ToolMessage	Phản hồi từ công cụ (tool)	Khi một tool được AI gọi và trả về kết quả
ToolCall	(Không phải message) – Gọi hàm/tool	Khi AI quyết định gọi tool thay vì trả lời

Tác giả: Đỗ Ngọc Tú
Công Ty Phần Mềm VHTSoft

Assistant và Agent

I. Assistant là gì?

Assistant trong thường là một hệ thống AI đơn lẻ được thiết kế để thực hiện các nhiệm vụ cụ thể, thường làm việc theo **prompt cố định**: ví dụ nhận input, trả lời output, không linh hoạt nhiều.

Trong nhiều dự án, bạn thấy "Assistant" = "OpenAI Assistant", tức là chỉ một chatbot đơn giản trả lời tin nhắn.

- **Đặc điểm:**
 - Thường là single-purpose (mục đích đơn lẻ)
 - Có phạm vi hoạt động hẹp và xác định rõ
 - Hoạt động theo luồng cố định (fixed workflow)
 - Ít khả năng ra quyết định phức tạp
- **Ví dụ:** Một assistant trả lời FAQ, assistant lập lịch họp, assistant dịch thuật

Dùng Assistant khi:

- Bạn cần giải pháp nhanh cho vấn đề đơn giản
- Workflow cố định và predictable (có thể dự đoán)
- Không cần tích hợp nhiều công cụ bên ngoài

Kiến trúc Assistant:

Assistant có kiến trúc rất đơn giản,

User Input → [Assistant] → Output

Ví dụ

- Bạn hỏi: "Hôm nay thời tiết thế nào?"
- Nó trả lời ngay: "Hôm nay trời nắng tại Hà Nội."

II. Agent là gì

Agent là một khái niệm nâng cao hơn với nhiều khả năng phức tạp:

- **Đặc điểm:**
 - Có agency (khả năng tự quyết định hành động)
 - Có thể sử dụng tools (công cụ bên ngoài)
 - Có trạng thái (state) được duy trì
 - Có thể học và thích nghi
 - Thường có feedback loop (vòng lặp phản hồi)

- **Ví dụ:** Agent nghiên cứu tự động, agent giao dịch chứng khoán, agent hỗ trợ phát triển phần mềm

Agent là một **thực thể** (entity) có thể:

- **Nhận đầu vào** (input),
- **Xử lý logic riêng**,
- **Quyết định hành động** (ví dụ: gọi API, chọn công cụ, tính toán, suy luận, trả lời),
- Và **trả ra đầu ra** (output).

Agent thường **tự động** làm việc, có **trí thông minh riêng** tùy theo mục đích lập trình cho nó thế nào.

Ví dụ Agent:

- Bạn hỏi: "Hôm nay thời tiết thế nào?"
- Nó suy nghĩ: "Tôi nên tra cứu API thời tiết → lấy dữ liệu → xử lý → trả lời."
- Thậm chí nếu lỗi API, nó có thể **chọn** hỏi lại bạn địa điểm, hoặc thử cách khác.

Tác giả: Đỗ Ngọc Tú
Công Ty Phần Mềm VHTSoft

HumanMessage – Tin nhắn từ người dùng

Trong LangChain, `HumanMessage` là một trong những loại message cơ bản nhất, đại diện cho **lời nói, câu hỏi hoặc yêu cầu từ người dùng** trong cuộc trò chuyện với mô hình ngôn ngữ (LLM).

Khi bạn gọi mô hình bằng `ChatModel` như `ChatOpenAI`, bạn có thể truyền vào một danh sách các message. `HumanMessage` mô phỏng nội dung do con người nhập vào — tương đương với việc bạn gõ câu hỏi vào ChatGPT.

Cú pháp khai báo

```
from langchain.schema import HumanMessage

message = HumanMessage(content="Hôm nay thời tiết thế nào?")
```

- `content`: là văn bản người dùng gửi.
- Có thể dùng thêm `name` nếu bạn muốn đặt tên cho người gửi (không bắt buộc).

Ví dụ đơn giản sử dụng với `ChatOpenAI`

```
from langchain.chat_models import ChatOpenAI
from langchain.schema import HumanMessage

# Khởi tạo mô hình
llm = ChatOpenAI()

# Tạo HumanMessage
user_message = HumanMessage(content="Hãy tính giúp tôi 7 + 5 là bao nhiêu?")

# Gửi message đến mô hình và in kết quả
response = llm([user_message])
print(response.content)
```

Kết quả có thể là:

```
7 + 5 = 12.
```

Dùng nhiều messages trong cuộc hội thoại

```
from langchain.schema import HumanMessage, AIMessage, SystemMessage
from langchain.chat_models import ChatOpenAI

chat = ChatOpenAI()

messages = [
    SystemMessage(content="Bạn là một trợ lý AI thông minh và lịch sự."),
    HumanMessage(content="Bạn tên là gì?"),
    AIMessage(content="Tôi là trợ lý AI được thiết kế để hỗ trợ bạn."),
    HumanMessage(content="Bạn có thể dịch câu 'Hello' sang tiếng Việt không?")
]

response = chat(messages)
print(response.content)
```

Tác giả: Đỗ Ngọc Tú
Công Ty Phần Mềm VHTSoft

AIMessage – Tin nhắn phản hồi từ AI

Trong LangChain, `AIMessage` đại diện cho **phản hồi từ mô hình ngôn ngữ** (LLM) sau khi nhận một hoặc nhiều `HumanMessage`.

Bạn có thể nghĩ nó giống như câu trả lời mà ChatGPT gửi lại sau khi bạn hỏi.

LangChain cho phép bạn truyền danh sách các message vào mô hình — trong đó `AIMessage` giúp mô phỏng những phản hồi trước đó từ AI để giữ **ngữ cảnh hội thoại**.

Cú pháp khai báo

```
from langchain.schema import AIMessage

message = AIMessage(content="Tôi có thể giúp gì cho bạn hôm nay?")
```

- `content`: là nội dung phản hồi của AI.
- Cũng có thể dùng thêm `name`, nhưng thường không cần vì AI chỉ có 1 vai trò chính.

Ví dụ sử dụng với `ChatOpenAI`

```
from langchain.chat_models import ChatOpenAI
from langchain.schema import HumanMessage, AIMessage

chat = ChatOpenAI()

messages = [
    HumanMessage(content="Bạn tên là gì?"),
    AIMessage(content="Tôi là trợ lý ảo của bạn."),
    HumanMessage(content="Bạn có thể giúp tôi dịch từ 'apple' sang tiếng Việt không?")
]

response = chat(messages)
print(response.content)
```

Kết quả có thể là:

"Từ 'apple' trong tiếng Việt là 'quả táo'."

Sử dụng trong hội thoại nhiều lượt (multi-turn conversation)

```
from langchain.schema import HumanMessage, AIMessage, SystemMessage

messages = [
    SystemMessage(content="Bạn là trợ lý lịch sự, trả lời bằng tiếng Việt."),
    HumanMessage(content="Xin chào!"),
    AIMessage(content="Chào bạn! Tôi có thể giúp gì hôm nay?"),
    HumanMessage(content="Thời tiết hôm nay như thế nào ở Hà Nội?")
]
```

Bạn có thể gửi `messages` này vào `ChatOpenAI()` để giữ toàn bộ ngữ cảnh từ đầu đến cuối.

`AIMessage` sử dụng trong những trường hợp

- Khi bạn **lưu trữ phản hồi của AI** trong một cuộc trò chuyện và muốn **phát lại** sau này.
- Khi bạn cần **thêm bối cảnh** vào lời nhắc (prompt) hiện tại để mô hình hiểu mạch hội thoại.
- Khi bạn muốn mô phỏng lại các cuộc đối thoại giữa người dùng và AI.

Tác giả: Đỗ Ngọc Tú
Công Ty Phần Mềm VHTSoft

SystemMessage - Thiết lập ngữ cảnh

Trong LangChain, `SystemMessage` là một loại message đặc biệt dùng để **thiết lập ngữ cảnh hoặc hướng dẫn hành vi** cho mô hình ngôn ngữ (LLM).

“ Nó giống như "lệnh nền" mà người dùng không thấy, nhưng ảnh hưởng đến cách AI trả lời.

Cú pháp

```
from langchain.schema import SystemMessage
```

```
system_msg = SystemMessage(content="Bạn là một trợ lý AI nói tiếng Việt và luôn lịch sự.")
```

- `content`: Văn bản chỉ dẫn cho mô hình.
- Không hiển thị với người dùng cuối, nhưng cực kỳ quan trọng để **định hình cá tính và hành vi** của AI.

Mục đích sử dụng

- Thiết lập **vai trò** cho AI (ví dụ: giáo viên, chuyên gia, chatbot vui vẻ...).
- Quy định **giọng điệu** (ví dụ: lịch sự, vui vẻ, nghiêm túc).
- Áp đặt **giới hạn**, như "không trả lời nếu không chắc chắn".

Ví dụ đơn giản

```
from langchain.chat_models import ChatOpenAI
from langchain.schema import HumanMessage, SystemMessage

llm = ChatOpenAI()

messages = [
    SystemMessage(content="Bạn là một trợ lý AI lịch sự và vui tính."),
    HumanMessage(content="Hôm nay bạn khỏe không?")
]
```

```
]
```

```
response = llm(messages)
print(response.content)
```

Kết quả có thể là

Cảm ơn bạn đã hỏi! Tôi là AI nên không có cảm giác, nhưng nếu có thì chắc chắn hôm nay tôi rất tuyệt vời. Còn bạn thì sao?

Thấy không? AI trở nên "vui tính" và "lịch sự" đúng như chỉ dẫn trong `SystemMessage`.

Ví dụ hội thoại có nhiều loại message

```
from langchain.schema import HumanMessage, AIMessage, SystemMessage
from langchain.chat_models import ChatOpenAI

llm = ChatOpenAI()

messages = [
    SystemMessage(content="Bạn là một chuyên gia lịch sử Việt Nam, nói ngắn gọn."),
    HumanMessage(content="Ai là vua đầu tiên của triều Nguyễn?"),
]

response = llm(messages)
print(response.content)
```

Kết quả mẫu:

Vua đầu tiên của triều Nguyễn là Gia Long, lên ngôi năm 1802.

Mẹo chuyên sâu

- `SystemMessage` nên luôn được **đặt đầu tiên** trong danh sách messages.
- Nếu dùng `memory`, bạn có thể **thêm `SystemMessage` một lần duy nhất** để mô hình nhớ bối cảnh trong suốt cuộc trò chuyện.
- Bạn có thể sử dụng `SystemMessage` để **giới hạn độ dài, độ chính xác, hoặc hành vi không mong muốn** của AI.
- Muốn ví dụ sử dụng `SystemMessage` để tạo các kiểu AI khác nhau như:
 - Trợ lý thân thiện
 - Chuyên gia tài chính
 - Luật sư

- Dịch giả tự động

Tác giả: Đỗ Ngọc Tú
Công Ty Phần Mềm VHTSoft

ToolMessage - Kết quả phản hồi từ một công cụ (tool)

Trong LangChain, `ToolMessage` đại diện cho **kết quả phản hồi từ một công cụ (tool)** sau khi mô hình LLM gọi tool đó.

“ Nó giống như AI bảo: "Tôi cần dùng máy tính để tính $2 + 2$ ", sau đó bạn gửi lại kết quả "4" bằng `ToolMessage` .

Khi nào dùng `ToolMessage` ?

Khi bạn dùng `bind_tools()` để cho phép LLM gọi các hàm (tools), LangChain sẽ:

1. **LLM tạo một ToolCall** (ví dụ: gọi hàm `multiply(a=2, b=3)`).
2. Bạn thực thi hàm Python tương ứng.
3. Sau đó, bạn dùng `ToolMessage` để **truyền kết quả trả về của tool đó lại cho LLM**.

Cú pháp

```
from langchain_core.messages import ToolMessage

msg = ToolMessage(tool_call_id="abc123", content="Kết quả là 6")
```

Các tham số:

Tham số	Ý nghĩa
<code>tool_call_id</code>	ID của lời gọi hàm (do <code>ToolCallMessage</code> sinh ra)
<code>content</code>	Nội dung trả lời (kết quả của tool)

Ví dụ đầy đủ: Gọi hàm nhân trong Python

1. Định nghĩa hàm (tool)

```
def multiply(a: int, b: int) -> int:
    """Nhân hai số."""
    return a * b
```

2. Dùng LLM + tool

```
from langchain_core.messages import HumanMessage, ToolMessage
from langchain_openai import ChatOpenAI

chat = ChatOpenAI().bind_tools([multiply], parallel_tool_calls=False)
```

3. LLM đưa ra ToolCall

```
messages = [
    HumanMessage(content="Hãy nhân 4 với 5")
]

response = chat.invoke(messages)
tool_call = response.tool_calls[0]
print(tool_call) # --> thông tin về lời gọi hàm multiply
```

4. Gửi kết quả về lại cho LLM bằng ToolMessage

```
# Gọi hàm thủ công
result = multiply(**tool_call.args)

# Gửi kết quả về lại
followup = chat.invoke([
    response,
    ToolMessage(tool_call_id=tool_call.id, content=str(result))
])

print(followup.content) # ==> LLM phản hồi dựa trên kết quả tool
```

Quan trọng

- `ToolMessage` cực kỳ quan trọng khi bạn dùng nhiều bước logic có liên quan đến tool.
- `tool_call_id` phải khớp chính xác với `ToolCallMessage` từ LLM, để LLM biết bạn đang trả lời cho yêu cầu nào.
- Bạn có thể dùng nhiều `ToolMessage` nếu LLM gọi nhiều hàm cùng lúc (nếu `parallel_tool_calls=True`).

Tác giả: Đỗ Ngọc Tú
Công Ty Phần Mềm VHTSoft

ToolCall - Mô hình ngôn ngữ gọi đến một công cụ

Trong LangChain, `ToolCall` đại diện cho việc **mô hình ngôn ngữ gọi đến một công cụ (function)** để thực hiện một hành động cụ thể.

“Nói cách khác, khi bạn dạy AI cách gọi các hàm Python, thì `ToolCall` là cách mô hình nói: “Tôi muốn dùng hàm `add(5, 3)` nhé.”

Tại sao cần `ToolCall`?

Mô hình ngôn ngữ rất giỏi hiểu và tạo ra văn bản, nhưng lại **không giỏi tính toán, truy vấn API, hoặc thao tác với dữ liệu phức tạp**.

Khi cần làm những việc như vậy, mô hình sẽ **gọi một “tool”** (được định nghĩa trước bằng Python), thông qua một `ToolCall`.

Cấu trúc cơ bản của `ToolCall`

```
from langchain_core.messages import ToolCall

tool_call = ToolCall(
    name="multiply",
    args={"a": 5, "b": 3},
    id="call_123"
)
```

Các thành phần:

- `name`: tên hàm (tool) mà AI muốn gọi (giống với tên bạn đăng ký khi dùng `bind_tools`).
- `args`: tham số truyền vào cho hàm đó.
- `id`: mã định danh của tool call (thường là tự sinh).

Ví dụ đầy đủ: Sử dụng `ToolCall` với LLM

Bước 1: Định nghĩa công cụ

```
def add(a: int, b: int) -> int:
    return a + b
```

Bước 2: Bind tool vào LLM

```
from langchain_openai import ChatOpenAI

llm = ChatOpenAI(model="gpt-4o")
llm_with_tools = llm.bind_tools([add])
```

Bước 3: Gửi tin nhắn có yêu cầu tính toán

```
from langchain_core.messages import HumanMessage

messages = [HumanMessage(content="Hãy cộng 4 và 7 giúp tôi")]
response = llm_with_tools.invoke(messages)

print(response.tool_calls) # Đây chính là danh sách ToolCall
```

Output mẫu:

```
[
  ToolCall(
    name='add',
    args={'a': 4, 'b': 7},
    id='toolu_abc123'
  )
]
```

Tức là mô hình quyết định gọi hàm `add` với `a=4`, `b=7` thông qua ToolCall.

Tương tác hoàn chỉnh: Gọi Tool và trả lại kết quả

```
from langchain_core.messages import AIMessage, ToolMessage

# Gọi hàm add
tool_result = add(4, 7)
```

```
# Trả kết quả lại cho AI thông qua ToolMessage
final_response = llm_with_tools.invoke([
    HumanMessage(content="Hãy cộng 4 và 7"),
    AIMessage(tool_calls=[ToolCall(name="add", args={"a": 4, "b": 7}, id="call_1")]),
    ToolMessage(tool_call_id="call_1", content=str(tool_result))
])

print(final_response.content)
```

Output

Kết quả là 11!

Tóm tắt so sánh

Thành phần	Vai trò
<code>ToolCall</code>	Yêu cầu gọi hàm từ mô hình LLM
<code>ToolMessage</code>	Trả kết quả từ tool về để LLM tiếp tục xử lý
<code>bind_tools</code>	Cho LLM biết có thể gọi những hàm nào

Khi nào cần dùng `ToolCall`?

- Khi mô hình phải tính toán, xử lý logic cụ thể, hoặc truy vấn dữ liệu thật.
- Khi xây dựng chatbot có khả năng "hành động", như đặt lịch, gọi API, v.v.

Tác giả: Đỗ Ngọc Tú
Công Ty Phần Mềm VHTSoft