Add message history (memory)

The `RunnableWithMessageHistory` let us add message history to certain types of chains.

Specifically, it can be used for any Runnable that takes as input one of

- a sequence of `BaseMessage`
- a dict with a key that takes a sequence of `BaseMessage`
- a dict with a key that takes the latest message(s) as a string or sequence of `BaseMessage`, and a separate key that takes historical messages

And returns as output one of

- a string that can be treated as the contents of an `AIMessage`
- a sequence of `BaseMessage`

- a dict with a key that contains a sequence of `BaseMessage`

Let's take a look at some examples to see how it works.

Setup

We'll use Redis to store our chat message histories and Anthropic's claude-2 model so we'll need to install the following dependencies:

```
%pip install --upgrade --quiet  langchain
redis anthropic
```

Set your Anthropic API key:

```
import getpass
import os

os.environ["ANTHROPIC_API_KEY"] =
getpass.getpass()
```

Start a local Redis Stack server if we don't have an existing Redis deployment to connect to:

```
docker run -d -p 6379:6379 -p 8001:8001
redis/redis-stack:latest
```

```
REDIS_URL = "redis://localhost:6379/0"
```

LangSmith

LangSmith is especially useful for something like message history injection, where it can be hard to otherwise understand what the inputs are to various parts of the chain.

Note that LangSmith is not needed, but it is helpful. If you do want to use LangSmith, after you sign up at the link above, make sure to uncoment the below and set your environment variables to start logging traces:

```
# os.environ["LANGCHAIN_TRACING_V2"] = "true"
# os.environ["LANGCHAIN_API_KEY"] =
getpass.getpass()
```

Example: Dict input, message output

Let’s create a simple chain that takes a dict as input and returns a BaseMessage.

In this case the `"question"` key in the input represents our input message, and the `"history"` key is where our historical messages will be injected.

```
from typing import Optional

from
langchain_community.chat_message_histories
import RedisChatMessageHistory
from langchain_community.chat_models import
ChatAnthropic
from langchain_core.chat_history import
BaseChatMessageHistory
from langchain_core.prompts import
ChatPromptTemplate, MessagesPlaceholder
from langchain_core.runnables.history import
RunnableWithMessageHistory
```

```
prompt = ChatPromptTemplate.from_messages(
    [
        ("system", "You're an assistant who's
good at {ability}"),

MessagesPlaceholder(variable_name="history"),
        ("human", "{question}"),
    ]
)
```

```
chain = prompt | ChatAnthropic(model="claude-
2")
```

Adding message history

To add message history to our original chain we wrap it in the `RunnableWithMessageHistory` class.

Crucially, we also need to define a method that takes a session_id string and based on it returns a `BaseChatMessageHistory`. Given the same input, this method should return an equivalent output.

In this case we'll also want to specify `input_messages_key` (the key to be treated as the latest input message) and `history_messages_key` (the key to add historical messages to).

```
chain_with_history =
RunnableWithMessageHistory(
    chain,
    lambda session_id:
RedisChatMessageHistory(session_id,
url=REDIS_URL),
    input_messages_key="question",
    history_messages_key="history",
)
```

Invoking with config

Whenever we call our chain with message history, we need to include a config that contains the `session_id`

```
config={"configurable": {"session_id": "
<SESSION_ID>"}}
```

Given the same configuration, our chain should be pulling from the same chat message history.

```
chain_with_history.invoke(
    {"ability": "math", "question": "What
does cosine mean?"},
    config={"configurable": {"session_id":
"foobar"}},
)
```

```
AIMessage(content=' Cosine is one of the
basic trigonometric functions in mathematics.
It is defined as the ratio of the adjacent
side to the hypotenuse in a right
triangle.\n\nSome key properties and facts
about cosine:\n\n- It is denoted by cos(θ),
where θ is the angle in a right triangle.
\n\n- The cosine of an acute angle is always
```

```
positive. For angles greater than 90 degrees, cosine can be negative.\n\n- Cosine is one of the three main trig functions along with sine and tangent.\n\n- The cosine of 0 degrees is 1. As the angle increases towards 90 degrees, the cosine value decreases towards 0.\n\n- The range of values for cosine is -1 to 1.\n\n- The cosine function maps angles in a circle to the x-coordinate on the unit circle.\n\n- Cosine is used to find adjacent side lengths in right triangles, and has many other applications in mathematics, physics, engineering and more.\n\n- Key cosine identities include: cos(A+B) = cosAcosB − sinAsinB and cos(2A) = cos^2(A) − sin^2(A)\n\nSo in summary, cosine is a fundamental trig')
```

```
chain_with_history.invoke(
    {"ability": "math", "question": "What's its inverse"},
    config={"configurable": {"session_id": "foobar"}},
)
```

```
AIMessage(content=' The inverse of the cosine function is called the arccosine or inverse cosine, often denoted as cos-1(x) or
```

```
arccos(x).\n\nThe key properties and facts
about arccosine:\n\n- It is defined as the
angle θ between 0 and π radians whose cosine
is x. So arccos(x) = θ such that cos(θ) =
x.\n\n- The range of arccosine is 0 to π
radians (0 to 180 degrees).\n\n- The domain
of arccosine is -1 to 1. \n\n- arccos(cos(θ))
= θ for values of θ from 0 to π radians.\n\n-
arccos(x) is the angle in a right triangle
whose adjacent side is x and hypotenuse is
1.\n\n- arccos(0) = 90 degrees. As x
increases from 0 to 1, arccos(x) decreases
from 90 to 0 degrees.\n\n- arccos(1) = 0
degrees. arccos(-1) = 180 degrees.\n\n- The
graph of y = arccos(x) is part of the unit
circle, restricted to x')
```

Langsmith trace

Looking at the Langsmith trace for the second call, we can see that when constructing the prompt, a “history” variable has been injected which is a list of two messages (our first input and first output).

Example: messages input, dict output

```
from langchain_core.messages import
HumanMessage
from langchain_core.runnables import
RunnableParallel

chain = RunnableParallel({"output_message":
ChatAnthropic(model="claude-2")})
chain_with_history =
RunnableWithMessageHistory(
    chain,
    lambda session_id:
RedisChatMessageHistory(session_id,
url=REDIS_URL),
    output_messages_key="output_message",
)

chain_with_history.invoke(
    [HumanMessage(content="What did Simone de
Beauvoir believe about free will")],
    config={"configurable": {"session_id":
"baz"}},
)
```

```
{'output_message': AIMessage(content=' Here
is a summary of Simone de Beauvoir\'s views
on free will:\n\n- De Beauvoir was an
existentialist philosopher and believed
strongly in the concept of free will. She
rejected the idea that human nature or
```

```
instincts determine behavior.\n\n- Instead, de Beauvoir argued that human beings define their own essence or nature through their actions and choices. As she famously wrote, "One is not born, but rather becomes, a woman."\n\n- De Beauvoir believed that while individuals are situated in certain cultural contexts and social conditions, they still have agency and the ability to transcend these situations. Freedom comes from choosing one\'s attitude toward these constraints.\n\n- She emphasized the radical freedom and responsibility of the individual. We are "condemned to be free" because we cannot escape making choices and taking responsibility for our choices. \n\n- De Beauvoir felt that many people evade their freedom and responsibility by adopting rigid mindsets, ideologies, or conforming uncritically to social roles.\n\n- She advocated for the recognition of ambiguity in the human condition and warned against the quest for absolute rules that deny freedom and responsibility. Authentic living involves embracing ambiguity.\n\nIn summary, de Beauvoir promoted an existential ethics')}
```

```
chain_with_history.invoke(
    [HumanMessage(content="How did this compare to Sartre")],
```

```
    config={"configurable": {"session_id":
"baz"}},
)
```

```
{'output_message': AIMessage(content=" There
are many similarities between Simone de
Beauvoir's views on free will and those of
Jean-Paul Sartre, though some key differences
emerge as well:\n\nSimilarities with
Sartre:\n\n- Both were existentialist
thinkers who rejected determinism and
emphasized human freedom and
responsibility.\n\n- They agreed that
existence precedes essence - there is no
predefined human nature that determines who
we are.\n\n- Individuals must define
themselves through their choices and actions.
This leads to anxiety but also freedom.\n\n-
The human condition is characterized by
ambiguity and uncertainty, rather than fixed
meanings/values.\n\n- Both felt that most
people evade their freedom through self-
deception, conformity, or adopting collective
identities/values
uncritically.\n\nDifferences from Sartre:
\n\n- Sartre placed more emphasis on the
burden and anguish of radical freedom. De
Beauvoir focused more on its positive
potential.\n\n- De Beauvoir critiqued
Sartre's premise that human relations are
```

```
necessarily conflictual. She saw more
potential for mutual recognition.\n\n- Sartre
saw the Other's gaze as a threat to freedom.
De Beauvoir put more stress on how the
Other's gaze can confirm")}
```

LangSmith trace

More examples

We could also do any of the below:

```
from operator import itemgetter

# messages in, messages out
RunnableWithMessageHistory(
    ChatAnthropic(model="claude-2"),
    lambda session_id:
RedisChatMessageHistory(session_id,
url=REDIS_URL),
)

# dict with single key for all messages in,
messages out
RunnableWithMessageHistory(
    itemgetter("input_messages") |
ChatAnthropic(model="claude-2"),
    lambda session_id:
```

```
RedisChatMessageHistory(session_id,
url=REDIS_URL),
    input_messages_key="input_messages",
)
```