以下是根据图片左侧内容整理的 **LangChain Memory 模块完整技术文档**：

LangChain Memory 核心解析

1. 概述与核心概念

- **功能定位**：维护对话状态，将历史信息注入未来对话
- **应用场景**：聊天机器人、多轮对话系统、长期交互应用
- **核心抽象类**：`BaseMemory`（定义统一接口）

```
from langchain_core.memory import BaseMemory

class BaseMemory(ABC):
    @property
    @abstractmethod
    def memory_variables(self) -> List[str]:  # 返回内存变量名

    @abstractmethod
    def load_memory_variables(self, inputs: Dict[str, Any]) -> Dict[str, Any]:
 # 加载内存

    @abstractmethod
    def save_context(self, inputs: Dict[str, Any], outputs: Dict[str, Any]):  #
保存上下文

    @abstractmethod
    def clear(self):  # 清除内存
```

---

2. Memory 核心类型

(1) `ConversationBufferMemory`

- **功能**：存储完整对话历史
- **特点**：
  - 保存所有 `HumanMessage` 和 `AIMessage`
  - 支持原始消息或格式化字符串
- **实现文件**：`libs/langchain/langchain/memory/buffer.py (1-30行)`

```
from langchain.memory import ConversationBufferMemory

memory = ConversationBufferMemory(
    memory_key="history",   # 存储键名
    human_prefix="User",    # 用户消息前缀
    ai_prefix="Assistant"   # AI消息前缀
)
```

(2) `ConversationStringBufferMemory`

- **功能**：专为字符串对话设计
- **特点**：
  - 存储格式化字符串（非消息对象）
  - 更适合普通语言模型
- **代码位置**：`buffer.py (91-120行)`

(3) `ConversationBufferWindowMemory`

- **功能**：仅保留最近对话片段
- **核心参数**：

```
memory = ConversationBufferWindowMemory(
    k=3  # 保留最近3轮对话
)
```

- **实现原理**：

```
def load_memory_variables(self, inputs):
    return {self.memory_key: self.buffer[-self.k :]}  # 切片取最近k条
```

(4) `ConversationSummaryMemory`

- **功能**：生成对话摘要（适用于长对话）
- **工作原理**：

```
def save_context(self, inputs, outputs):
    self.buffer.append(...)  # 保存新对话
    if len(self.buffer) > self.max_token_limit:
        summary = self.llm.predict(self._get_prompt())  # 调用LLM生成摘要
        self.buffer = [summary]  # 替换为摘要
```

---

3. 基础使用方法

单链集成

```
from langchain.chains import ConversationChain
from langchain_community.llms import OpenAI

conversation = ConversationChain(
    llm=OpenAI(),
    memory=ConversationBufferMemory()
)

# 进行多轮对话
conversation("你好，我是小明")  # 第一轮
conversation("我刚才说了什么？")  # 第二轮（包含历史）
```

自定义 Memory

```
class CustomMemory(BaseMemory):
    memories: Dict[str, Any] = {}  # 自定义存储

    @property
    def memory_variables(self) -> List[str]:
        return list(self.memories.keys())  # 返回所有键

    def load_memory_variables(self, inputs):
        return self.memories  # 返回完整内存
```

---

4. 高级用法

(1) 与 `RunnableWithMessageHistory` 集成（推荐）

```
from langchain_core.runnables.history import RunnableWithMessageHistory
from langchain_core.chat_history import InMemoryChatMessageHistory
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.messages import HumanMessage, AIMessage

from langchain_ollama import OllamaLLM
llm = OllamaLLM(
    model="llama3.2",  # 指定使用的模型名称，或 "mistral", "qwen" 等
    temperature=0.3,   # 控制回答的创造性，值越低回答越确定
    num_gpu=1,         # 启用 GPU 加速，使用1个GPU
    num_thread=8,      # 设置 CPU 线程数为8
    system="你是一个专业的文档助手，回答要基于提供的上下文"  # 系统角色设定
 )
store = {}

def get_session_history(session_id : str ):
    if session_id not in store:
        store[session_id] = InMemoryChatMessageHistory()
    return store[session_id]

# 创建提示模板，显式定义用户消息角色
prompt = ChatPromptTemplate.from_messages([
    ("human", "{input}")
])

# 使用提示模板包装LLM，确保消息格式正确
llm_chain = prompt | llm

# 初始化带消息历史的链条，添加必要的消息键配置
chain = RunnableWithMessageHistory(
    llm_chain,
    get_session_history,
    input_messages_key="input",  # 指定用户输入键
    output_messages_key="output"   # 指定模型输出键
)

result01 = chain.invoke(
```

```
    {"input": "你好，我是小明"},  # 使用字典格式传递input键
    config={"configurable": {"session_id": "user_123"}}
)

print(result01)
print("*******************************")
print(store)
result02 = chain.invoke(
    {"input": "你好，我是小芳"},  # 同样使用字典格式
    config={"configurable": {"session_id": "user_123"}}
)
print("*******************************")
print(result02)
print("*******************************")
print(store)
```

(2) 内存共享

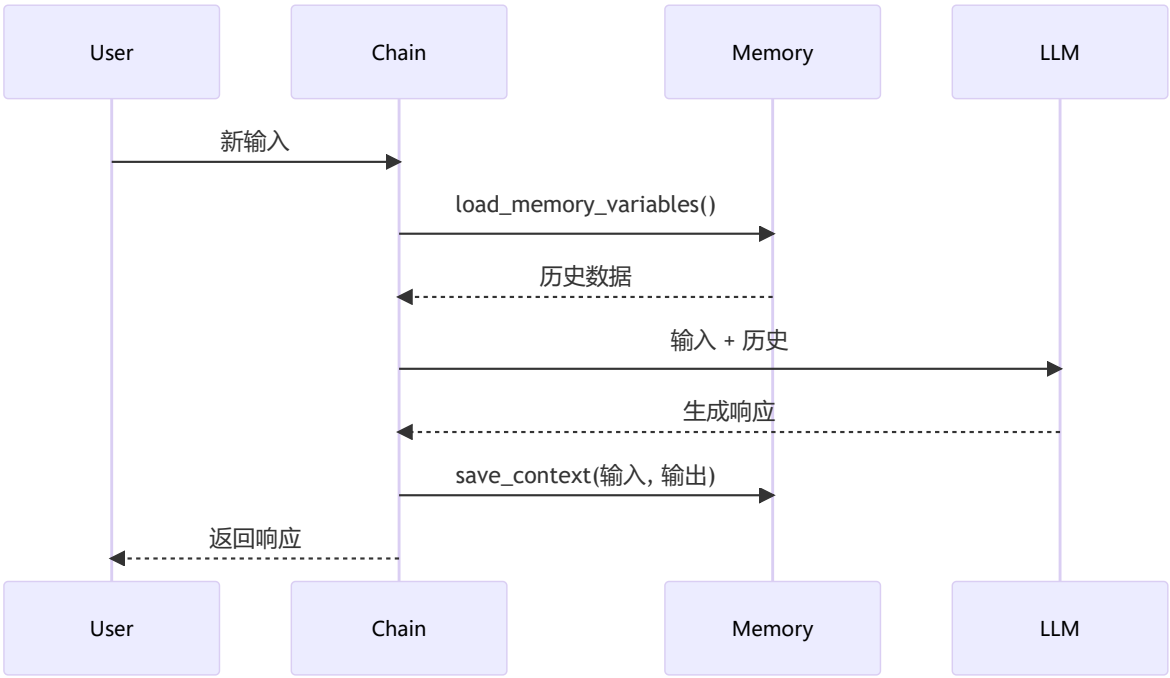
```
from langchain.memory import ReadOnlySharedMemory

base_memory = ConversationBufferMemory()
read_only_memory = ReadOnlySharedMemory(memory=base_memory)  # 创建只读副本
```

(3) Remembrall 长期记忆集成

```
chat_model = ChatOpenAI(
    openai_api_base="https://...",
    model_kwargs={"memory_type": "remembrall"}  # 启用长期记忆
)
```

---

5. 内存工作原理



生命周期阶段：

1. **初始化**：创建内存对象
2. **加载**：`load_memory_variables()` 注入历史
3. **使用**：整合到提示词影响LLM输出
4. **保存**：`save_context()` 存储新对话
5. **清除**：`clear()` 重置内存

---

6. 迁移指南

从旧版迁移

```
# 旧版（不推荐）
from langchain.chains import ConversationChain
conversation = ConversationChain(llm=OpenAI())

# 新版（推荐）
from langchain_core.runnables.history import RunnableWithMessageHistory
chain = RunnableWithMessageHistory(llm, get_session_history)
```

内存类型变更

```
# 从完整存储改为窗口存储
memory = ConversationBufferMemory()  # 旧
memory = ConversationBufferWindowMemory(k=3)  # 新（只保留3轮）
```

---

7. 选型指南

内存类型	适用场景	优点	限制
`BufferMemory`	短对话、调试	完整历史记录	上下文过长
`BufferWindowMemory`	通用场景	控制上下文长度	丢失早期信息
`SummaryMemory`	长对话（>10轮）	压缩关键信息	摘要可能失真
`Rembrall`	长期记忆需求	检索增强生成	需要额外存储

> **最佳实践**：
>
> 1. 普通聊天机器人 → `BufferWindowMemory(k=5)`
> 2. 客服系统 → `SummaryMemory + Rembrall`
> 3. 多会话应用 → `RunnableWithMessageHistory + InMemoryChatMessagestory`