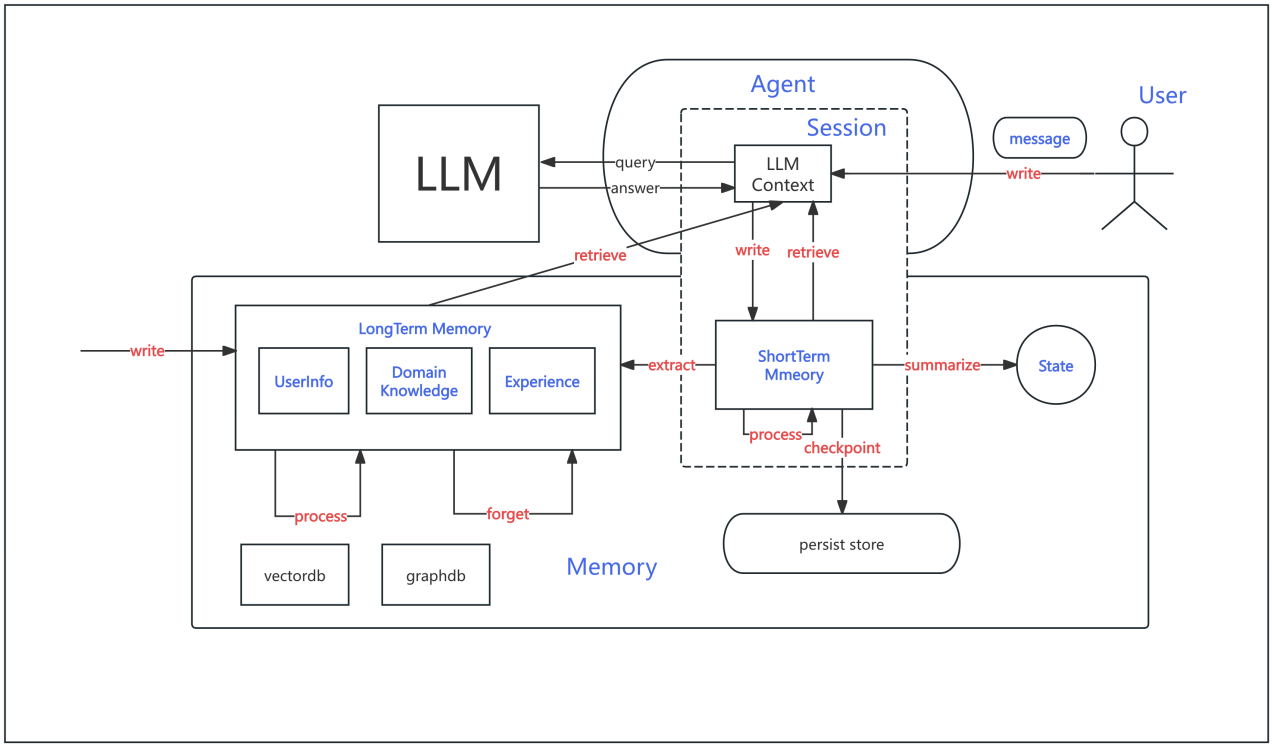
KeMem 详细设计

1. 用户代码Demo

```
from kemem import Memory

# 初始化
memory = Memory()
client = OpenAI()
USER_ID = "user_001"

def chat(user_message):
    """处理聊天消息"""
    print(f"\n用户: {user_message}")

    # 1. 从 kemem 搜索相关记忆
    memories = memory.search(user_message, user_id=USER_ID, limit=5)

    # 2. 构建提示词（包含记忆上下文）
    system_prompt = f"你是一个友好的助手。"
    system_prompt += f"\n\n关于用户的记忆：\n{memories}"

    # 3. 调用大语言模型
    response = client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[
            {"role": "system", "content": system_prompt},
            {"role": "user", "content": user_message}
        ]
    )

    # 4. 获取大语言模型的回复
    assistant_reply = response.choices[0].message.content
    print(f"助手: {assistant_reply}")

    # 5. 将对话存入 kemem
    memory.add([
        {"role": "user", "content": user_message},
        {"role": "assistant", "content": assistant_reply}
    ], user_id=USER_ID)

    return assistant_reply
```

2. 核心类设计



核心类说明

- **Agent** KeMem系统可以接入 多个 Agent
- **User** 一个Agent 可以有多个 User
- **Session** 一个 User 可以创建多个 Session，代表与 Agent 的一轮交互
- **Message** 一轮交互中有多个 Message。Message 通过 <AgentId, UserId, SessionId, CreateTime> 可以唯一确定
- **Memory** 代表某个 Agent 的全部记忆
  - **ShortTermMemory** 与 Session 一一对应，当交互结束时，数据会丢失
  - **LongTermMemory** 数据持久保存，不会丢失
    - **UserInfo** 用户的个性化数据，可以基于此数据，对Agent用户做个性化的任务生成与执行
    - **Domain Knowledge** 领域知识数据，被Agent 所有用户所共享
    - **Experience** 经验数据，由Memory系统自动从ShortTermMemory总结出来，并写入LongTermMemory
- **State** 当前hortTermMemory中信息的分层摘要，在发给大模型推理时用来减少 token 数量

类 Memory

1. 接口 write/add

```
函数原型：
def add(
    self,
    messages,
    user_id: Optional[str] = None,
    agent_id: Optional[str] = None,
    session_id: Optional[str] = None,
    metadata: Optional[Dict[str, Any]] = None,
    infer: bool = True,
    memory_type: Optional[str] = None,
    prompt: Optional[str] = None,
)

功能：添加新的记忆。通过 infer 的配置可以自动从对话中提取事实、智能合并/更新现有记忆。

参数:
messages (str | dict | list): 要存储的消息内容
可以是字符串: "用户喜欢编程"
可以是字典: {"role": "user", "content": "..."}
可以是消息列表: [{"role": "user", "content": "..."}, {"role": "assistant", "content"

user_id (str, optional): 用户标识符，用于隔离不同用户的记忆
agent_id (str, optional): 代理标识符，用于隔离不同代理的记忆
session_id (str, optional): 运行标识符，用于隔离不同运行会话的记忆
metadata (dict, optional): 附加元数据，可存储任意自定义信息
infer (bool): 是否启用智能推理模式（默认 True）
   True: LLM 自动提取事实并决定是添加、更新还是删除记忆
   False: 直接将消息原样存储为记忆
memory_type (str, optional): 记忆类型 (short/long)
prompt (str, optional): 自定义提示词，用于引导 LLM 提取记忆
```

2. 接口 extract

在调用add时被自动调用，提取用户当前 message 中的实体，关系，写入知识图谱

3. 接口 summarize

在调用add时被自动调用，将当前 ShortTermMemory 中的 Message 分层做摘要

4. 接口 get

函数原型：
```
def get(self, memory_id)
```

功能：根据 ID 检索单个记忆的详细信息

参数：
memory_id (str): 记忆的唯一标识符

5. 接口 get_all

函数原型：
```
def get_all(
    self,
    user_id: Optional[str] = None,
    agent_id: Optional[str] = None,
    session_id: Optional[str] = None,
    filters: Optional[Dict[str, Any]] = None,
    limit: int = 100,
)
```

功能：获取所有记忆列表，支持多维度过滤。此函数也可以用作获取最近的若干条 message

参数：
user_id (str, optional): 按用户ID过滤
agent_id (str, optional): 按代理ID过滤
session_id (str, optional): 按SessionID过滤
filters (dict, optional): 额外的自定义过滤条件
limit (int): 返回的最大记忆数量，默认 100

6. 接口 retrieve/search

函数原型：
```
def search(
    self,
    query: str,
    user_id: Optional[str] = None,
    agent_id: Optional[str] = None,
    run_id: Optional[str] = None,
    limit: int = 100,
    filters: Optional[Dict[str, Any]] = None,
    threshold: Optional[float] = None,
    rerank: bool = True,
```

```
)

功能: 基于语义相似度搜索记忆，支持重排序和高级过滤。

参数:
query (str): 搜索查询文本
user_id (str, optional): 限定搜索范围到特定用户
agent_id (str, optional): 限定搜索范围到特定代理
session_id (str, optional): 限定搜索范围到特定运行
limit (int): 返回结果数量上限，默认 100
threshold (float, optional): 相似度阈值 (0-1)，只返回相似度 >= 此值的结果
rerank (bool): 是否使用重排序模型优化结果顺序，默认 True
```

7. 接口 forget/delete

```
函数原型:
def delete(self, memory_id)

功能: 删除指定ID的单个记忆。除了用户主动调用外，还支持根据一定的策略自动调用，以忘记过期的，脏的记

参数:
memory_id (str): 要删除的记忆ID
```

8. 接口 delete_all()

```
函数原型:
def delete_all(
    self,
    user_id: Optional[str] = None,
    agent_id: Optional[str] = None,
    session_id: Optional[str] = None,
    filters: Optional[Dict[str, Any]] = None
)

功能: 批量删除符合条件的所有记忆。危险操作!

参数:
user_id (str, optional): 删除指定用户的所有记忆
agent_id (str, optional): 删除指定代理的所有记忆
session_id (str, optional): 删除指定运行的所有记忆
filters: 过滤条件
```

9. 接口 process

基于用户场景，自定义的数据处理算法

10. 接口 checkpoint

短期记忆可以通过checkpoint，定期保存记忆快照。对于长期运行的 Agent 任务，当系统崩溃时，可以通过

类 Table / KV / Document

用于对接不同的存储系统

类 LLM

用于对接不同的大语言模型。在信息提取时会使用

类 Embedding

用于对接不同的embedding模型。在信息提取，信息检索时会使用

类 VectorDB

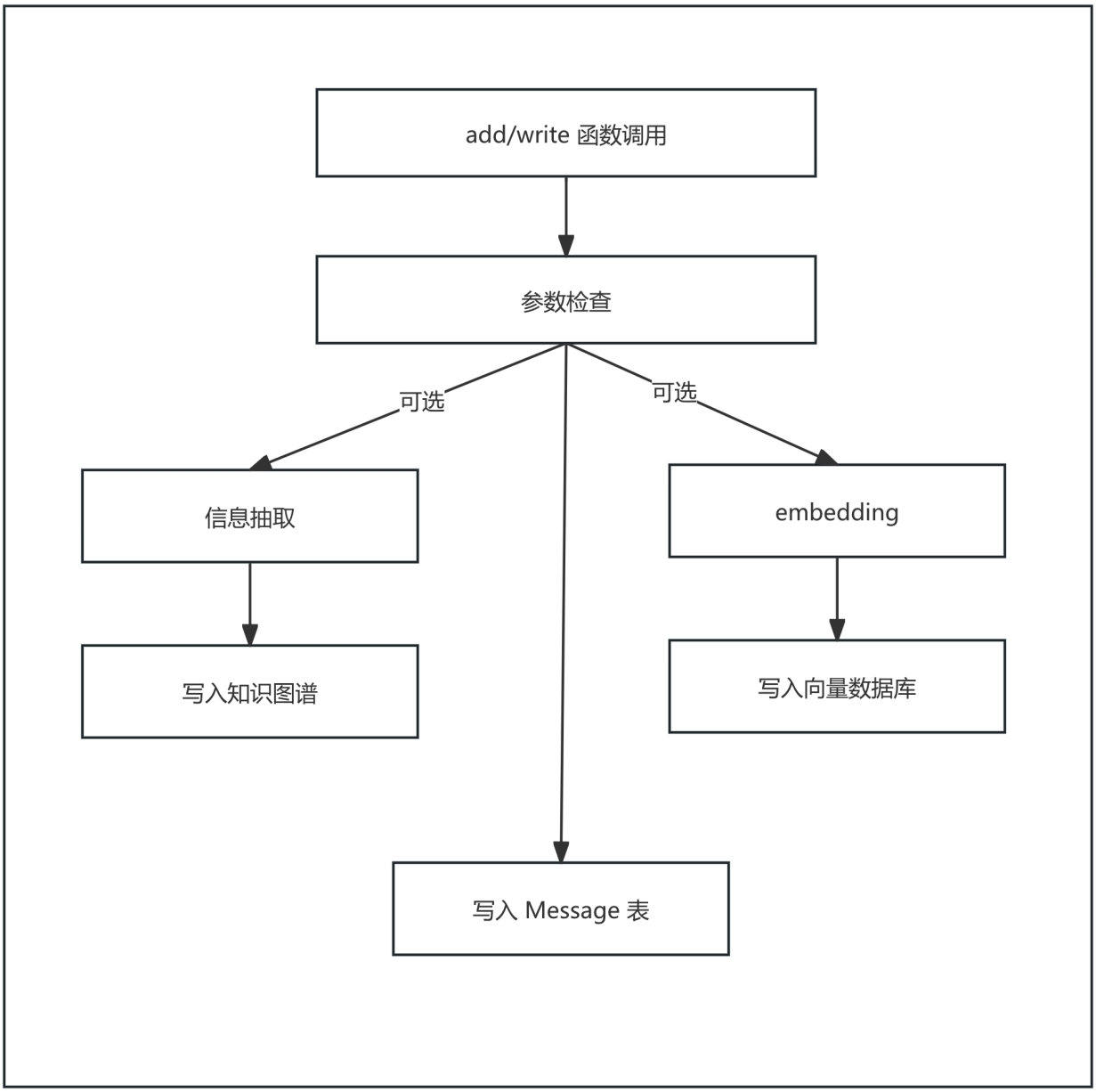
用于对接不同的向量数据库，在信息提取，信息检索时会使用

类 GraphDB

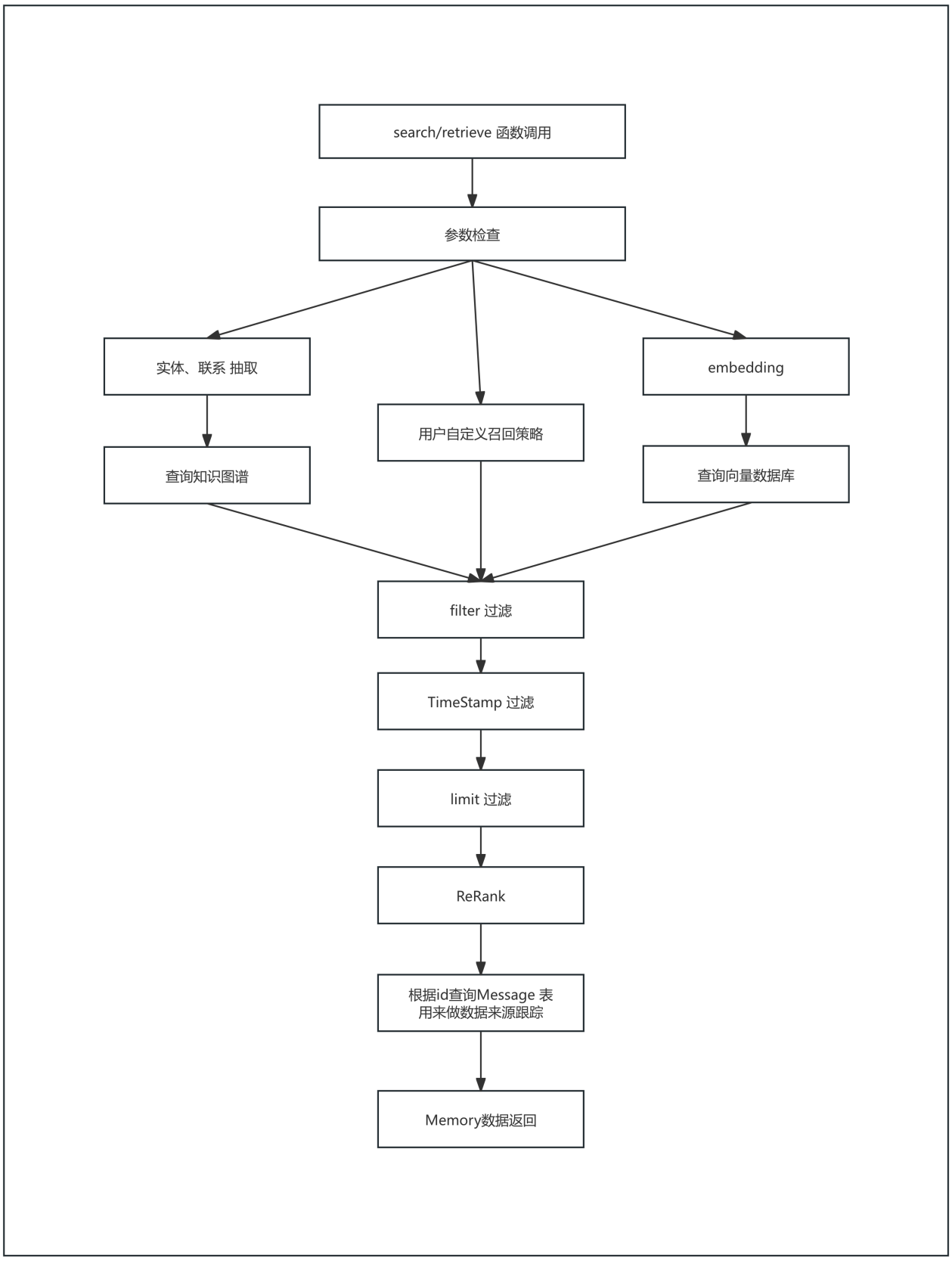
用于对接不同的图数据库，在信息提取，信息检索时会使用

3. 核心函数处理流程

3.1 write/add



3.2 retrieve/search



实体、联系抽取的例子

```
messages = [
    {
        "role": "user",
        "content": "我叫李明，在北京大学学习计算机科学。我的导师是张教授，他在人工智能领域非常
```

```
    }
]

提取实体(通过 LLM)
entity_type_map = {
    "李明": "person",           # 人名
    "北京大学": "organization",  # 组织/机构
    "计算机科学": "field",       # 学科领域
    "张教授": "person",          # 人名
    "人工智能": "field"          # 学科领域
}

建立关系(通过 LLM)
entities = [
    {
        "source": "李明",
        "relationship": "studies_at",
        "destination": "北京大学"
    },
    {
        "source": "李明",
        "relationship": "studies",
        "destination": "计算机科学"
    },
    {
        "source": "李明",
        "relationship": "advised_by",
        "destination": "张教授"
    },
    {
        "source": "张教授",
        "relationship": "expert_in",
        "destination": "人工智能"
    }
]

建立知识图谱，写入图数据库
李明 --[studies_at]--> 北京大学
李明 --[studies]--> 计算机科学
李明 --[advised_by]--> 张教授
张教授 --[expert_in]--> 人工智能
```

4. 元信息表设计

4.1 Agent表

字段名	类型	说明	约束
agent_id	VARCHAR(64)	Agent唯一标识	PRIMARY KEY
agent_name	VARCHAR(255)	Agent名称	NOT NULL
agent_type	VARCHAR(50)	Agent类型（如：chatbot, assistant等）	
description	TEXT	Agent描述信息	
config	JSON	Agent配置信息（模型参数等）	
status	TINYINT	状态（0-禁用，1-启用）	DEFAULT 1
create_time	TIMESTAMP	创建时间	NOT NULL, DEFAULT CURRENT_TIMESTAMP
update_time	TIMESTAMP	更新时间	NOT NULL, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
creator	VARCHAR(64)	创建者	

索引：

- PRIMARY KEY: `agent_id`
- INDEX: `idx_status` (status)
- INDEX: `idx_create_time` (create_time)

4.2 User表

字段名	类型	说明	约束
user_id	VARCHAR(64)	用户唯一标识	PRIMARY KEY
agent_id	VARCHAR(64)	所属Agent ID	NOT NULL
user_name	VARCHAR(255)	用户名称	
user_type	VARCHAR(50)	用户类型（如：individual, enterprise等）	
profile	JSON	用户画像信息	
status	TINYINT	状态（0-禁用，1-启用）	DEFAULT 1
create_time	TIMESTAMP	创建时间	NOT NULL, DEFAULT CURRENT_TIMESTAMP
update_time	TIMESTAMP	更新时间	NOT NULL, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP

字段名	类型	说明	约束
last_active_time	TIMESTAMP	最后活跃时间	

索引：

- PRIMARY KEY: `user_id`
- INDEX: `idx_agent_id` (agent_id)
- INDEX: `idx_agent_status` (agent_id, status)
- INDEX: `idx_last_active_time` (last_active_time)

外键：

- FOREIGN KEY: `agent_id` REFERENCES Agent(agent_id)

4.3 Session表

字段名	类型	说明	约束
session_id	VARCHAR(64)	Session唯一标识	PRIMARY KEY
agent_id	VARCHAR(64)	所属Agent ID	NOT NULL
user_id	VARCHAR(64)	所属User ID	NOT NULL
session_name	VARCHAR(255)	Session名称	
session_type	VARCHAR(50)	Session类型（如：chat, task等）	
context	JSON	会话上下文信息	
status	TINYINT	状态（0-结束，1-进行中）	DEFAULT 1
create_time	TIMESTAMP	创建时间	NOT NULL, DEFAULT CURRENT_TIMESTAMP
update_time	TIMESTAMP	更新时间	NOT NULL, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
end_time	TIMESTAMP	结束时间	

索引：

- PRIMARY KEY: `session_id`
- INDEX: `idx_agent_user` (agent_id, user_id)
- INDEX: `idx_user_id` (user_id)

- INDEX: `idx_status_create_time` (status, create_time)
- INDEX: `idx_create_time` (create_time)

外键:

- FOREIGN KEY: `agent_id` REFERENCES Agent(agent_id)
- FOREIGN KEY: `user_id` REFERENCES User(user_id)

4.4 Message表

字段名	类型	说明	约束
message_id	VARCHAR(64)	Message唯一标识	PRIMARY KEY
agent_id	VARCHAR(64)	所属Agent ID	NOT NULL
user_id	VARCHAR(64)	所属User ID	NOT NULL
session_id	VARCHAR(64)	所属Session ID	NOT NULL
role	VARCHAR(20)	角色（user, assistant, system）	NOT NULL
content	TEXT	消息内容	NOT NULL
content_type	VARCHAR(50)	内容类型（text, image, audio, video等）	DEFAULT 'text'
metadata	JSON	消息元数据（tokens, model等）	
parent_message_id	VARCHAR(64)	父消息ID（用于消息链）	
create_time	TIMESTAMP(6)	创建时间（微秒精度）	NOT NULL, DEFAULT CURRENT_TIMESTAMP(6)
sequence_no	BIGINT	序列号（会话内递增）	NOT NULL

索引:

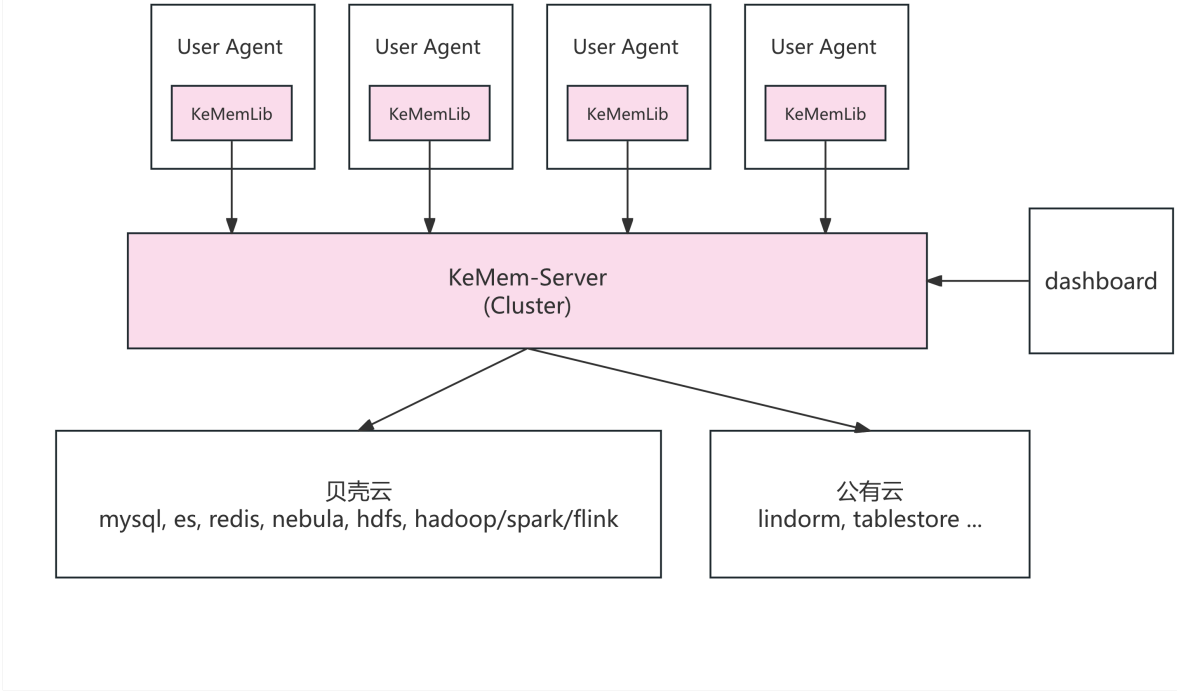
- PRIMARY KEY: `message_id`
- UNIQUE INDEX: `uk_agent_user_session_time` (agent_id, user_id, session_id, create_time)
- INDEX: `idx_session_create_time` (session_id, create_time)
- INDEX: `idx_session_sequence` (session_id, sequence_no)
- INDEX: `idx_user_create_time` (user_id, create_time)
- INDEX: `idx_create_time` (create_time)

外键：

- FOREIGN KEY: `agent_id` REFERENCES Agent(agent_id)
- FOREIGN KEY: `user_id` REFERENCES User(user_id)
- FOREIGN KEY: `session_id` REFERENCES Session(session_id)

> **说明：**
>
> - 通过 (agent_id, user_id, session_id, create_time) 建立唯一索引，满足业务需求
> - create_time 使用微秒精度（TIMESTAMP(6)），避免同一时刻多条消息冲突
> - sequence_no 作为会话内的递增序列号，便于按顺序检索消息
> - parent_message_id 支持构建消息树结构（如多轮对话分支）

5. 部署方案



6. 其它功能

- 支持异步接口，提高系统吞吐

- 支持 http 接口，方便调试
- 支持用户自定义process数据处理函数
- dashboard，用来做 数据监控，效果分析