**Personal AI Memory System - 使用Graphiti MCP + Notion SDK**

**快速开始**

**1. 环境安装**

bash

*# 克隆Graphiti*

git clone https://github.com/getzep/graphiti.git

cd graphiti

*# 安装依赖*

pip install notion-client *# 官方Python SDK*

uv sync *# 安装Graphiti依赖*

**2. 环境变量 (.env)**

bash

*# 在graphiti/mcp\_server/.env*

NOTION\_API\_KEY=your\_notion\_api\_key

OPENAI\_API\_KEY=your\_openai\_api\_key

NEO4J\_URI=bolt://localhost:7687

NEO4J\_USER=neo4j

NEO4J\_PASSWORD=password

**核心同步代码**

**1. Notion 数据提取器**

python

*# notion\_extractor.py*

from notion\_client import AsyncClient

from notion\_client.helpers import async\_collect\_paginated\_api

import re

from typing import List, Dict, Optional

class NotionExtractor:

def \_\_init\_\_(self, api\_key: str):

self.client = AsyncClient(auth=api\_key)

async def get\_all\_pages\_metadata(self) -> List[Dict]:

"""获取所有页面元数据"""

pages = await async\_collect\_paginated\_api(

self.client.search,

filter={"object": "page"}

)

metadata\_list = []

for page in pages:

metadata = await self.\_extract\_page\_metadata(page)

if metadata:

metadata\_list.append(metadata)

return metadata\_list

async def \_extract\_page\_metadata(self, page: Dict) -> Optional[Dict]:

"""提取页面元数据（不包含content）"""

try:

*# 获取内链和提及（只读取前10个block避免API限制）*

blocks = await self.client.blocks.children.list(

block\_id=page['id'],

page\_size=10

)

internal\_links = []

mentions = []

for block in blocks['results']:

text = self.\_extract\_text\_from\_block(block)

internal\_links.extend(re.findall(r'\[\[([^\]]+)\]\]', text))

mentions.extend(re.findall(r'@(\w+)', text))

return {

'notion\_id': page['id'],

'title': self.\_extract\_title(page),

'type': 'page',

'tags': self.\_extract\_tags(page),

'last\_edited\_time': page['last\_edited\_time'],

'url': page['url'],

'parent\_id': self.\_extract\_parent\_id(page),

'internal\_links': list(set(internal\_links)),

'mentions': list(set(mentions))

}

except Exception as e:

print(f"Error extracting metadata for page {page['id']}: {e}")

return None

def \_extract\_title(self, page: Dict) -> str:

"""提取页面标题"""

properties = page.get('properties', {})

for prop\_name, prop\_data in properties.items():

if prop\_data.get('type') == 'title':

title\_array = prop\_data.get('title', [])

if title\_array:

return ''.join([item.get('plain\_text', '') for item in title\_array])

return "Untitled"

def \_extract\_tags(self, page: Dict) -> List[str]:

"""提取标签"""

tags = []

properties = page.get('properties', {})

for prop\_data in properties.values():

if prop\_data.get('type') == 'multi\_select':

for option in prop\_data.get('multi\_select', []):

tags.append(option.get('name', ''))

elif prop\_data.get('type') == 'select':

select\_option = prop\_data.get('select')

if select\_option:

tags.append(select\_option.get('name', ''))

return [tag for tag in tags if tag]

def \_extract\_parent\_id(self, page: Dict) -> Optional[str]:

"""提取父页面ID"""

parent = page.get('parent', {})

if parent.get('type') == 'page\_id':

return parent.get('page\_id')

return None

def \_extract\_text\_from\_block(self, block: Dict) -> str:

"""从block中提取文本"""

block\_type = block.get('type', '')

if block\_type in ['paragraph', 'heading\_1', 'heading\_2', 'heading\_3', 'bulleted\_list\_item']:

block\_data = block.get(block\_type, {})

rich\_text = block\_data.get('rich\_text', [])

return ''.join([item.get('plain\_text', '') for item in rich\_text])

return ""

async def get\_page\_content(self, page\_id: str) -> str:

"""获取页面完整内容（用于最终检索）"""

try:

blocks = await async\_collect\_paginated\_api(

self.client.blocks.children.list,

block\_id=page\_id

)

content\_parts = []

for block in blocks:

text = self.\_extract\_text\_from\_block(block)

if text.strip():

content\_parts.append(text.strip())

return '\n'.join(content\_parts)

except Exception as e:

print(f"Error getting content for page {page\_id}: {e}")

return ""

**2. 扩展Graphiti MCP Server**

python

*# extended\_mcp\_server.py*

import asyncio

import os

from graphiti\_core import Graphiti

from notion\_extractor import NotionExtractor

from typing import Dict, List

class ExtendedGraphitiMCP:

def \_\_init\_\_(self):

self.graphiti = Graphiti(

neo4j\_uri=os.getenv('NEO4J\_URI', 'bolt://localhost:7687'),

neo4j\_user=os.getenv('NEO4J\_USER', 'neo4j'),

neo4j\_password=os.getenv('NEO4J\_PASSWORD', 'password')

)

self.notion = NotionExtractor(os.getenv('NOTION\_API\_KEY'))

self.synced\_pages = set()

async def sync\_notion\_to\_graph(self):

"""同步Notion到图数据库"""

print("Starting Notion sync...")

*# 获取所有页面元数据*

pages\_metadata = await self.notion.get\_all\_pages\_metadata()

print(f"Found {len(pages\_metadata)} pages")

*# 1. 添加页面节点*

for page\_meta in pages\_metadata:

await self.\_add\_page\_episode(page\_meta)

self.synced\_pages.add(page\_meta['notion\_id'])

*# 2. 创建关系*

await self.\_create\_relations(pages\_metadata)

print(f"Sync completed: {len(pages\_metadata)} pages processed")

async def \_add\_page\_episode(self, page\_meta: Dict):

"""添加页面episode"""

*# 创建摘要（用于embedding）*

summary = f"Title: {page\_meta['title']}"

if page\_meta['tags']:

summary += f", Tags: {', '.join(page\_meta['tags'])}"

await self.graphiti.add\_episode(

name=page\_meta['title'],

episode\_body=summary,

source\_description=f"notion\_page:{page\_meta['notion\_id']}"

)

async def \_create\_relations(self, pages\_metadata: List[Dict]):

"""创建五种关系"""

title\_to\_id = {page['title']: page['notion\_id'] for page in pages\_metadata}

for page in pages\_metadata:

relations = []

*# 1. CHILD\_OF关系*

if page['parent\_id']:

relations.append(f"CHILD\_OF:{page['parent\_id']}")

*# 2. LINKS\_TO关系*

for link\_title in page['internal\_links']:

if link\_title in title\_to\_id:

relations.append(f"LINKS\_TO:{title\_to\_id[link\_title]}")

*# 3. MENTIONS关系*

for mention in page['mentions']:

if mention in title\_to\_id:

relations.append(f"MENTIONS:{title\_to\_id[mention]}")

*# 4. HAS\_TAG关系*

for tag in page['tags']:

relations.append(f"HAS\_TAG:{tag}")

*# 创建关系episode*

if relations:

await self.graphiti.add\_episode(

name=f"Relations for {page['title']}",

episode\_body=f"Page {page['notion\_id']} has relations: {', '.join(relations)}",

source\_description=f"notion\_relations:{page['notion\_id']}"

)

async def search\_and\_retrieve(self, query: str, limit: int = 5) -> Dict:

"""搜索图谱并获取Notion内容"""

*# 1. 在图中搜索*

search\_results = await self.graphiti.search(query, num\_results=limit)

*# 2. 提取notion\_id*

notion\_ids = []

for edge in search\_results.edges:

source\_desc = edge.source\_node.name

*# 从source\_description中提取notion\_id*

if "notion\_page:" in source\_desc:

notion\_id = source\_desc.split("notion\_page:")[1].split()[0]

notion\_ids.append(notion\_id)

*# 3. 获取实际内容*

content\_results = {}

for notion\_id in notion\_ids[:limit]:

if notion\_id in self.synced\_pages:

content = await self.notion.get\_page\_content(notion\_id)

content\_results[notion\_id] = content

return {

'query': query,

'notion\_ids': notion\_ids,

'content': content\_results,

'graph\_results': len(search\_results.edges)

}

**3. 修改Graphiti MCP Server**

python

*# 在graphiti/mcp\_server/graphiti\_mcp\_server.py中添加自定义工具*

from extended\_mcp\_server import ExtendedGraphitiMCP

*# 在现有的MCP server中添加*

extended\_mcp = ExtendedGraphitiMCP()

@server.list\_tools()

async def handle\_list\_tools() -> list[types.Tool]:

"""添加自定义工具"""

base\_tools = [

*# ... 现有的Graphiti工具*

]

*# 添加Notion同步工具*

custom\_tools = [

types.Tool(

name="sync\_notion",

description="Sync Notion workspace to knowledge graph",

inputSchema={

"type": "object",

"properties": {},

"required": []

}

),

types.Tool(

name="search\_notion\_memory",

description="Search knowledge graph and retrieve Notion content",

inputSchema={

"type": "object",

"properties": {

"query": {"type": "string"},

"limit": {"type": "integer", "default": 5}

},

"required": ["query"]

}

)

]

return base\_tools + custom\_tools

@server.call\_tool()

async def handle\_call\_tool(name: str, arguments: dict) -> list[types.TextContent]:

"""处理工具调用"""

if name == "sync\_notion":

await extended\_mcp.sync\_notion\_to\_graph()

return [types.TextContent(type="text", text="Notion sync completed")]

elif name == "search\_notion\_memory":

query = arguments.get("query", "")

limit = arguments.get("limit", 5)

results = await extended\_mcp.search\_and\_retrieve(query, limit)

response = f"Found {results['graph\_results']} graph results for '{query}'\n\n"

for notion\_id, content in results['content'].items():

response += f"=== Page {notion\_id} ===\n"

response += content[:500] + "...\n\n"

return [types.TextContent(type="text", text=response)]

*# ... 处理其他工具*

**使用方法**

**1. 启动MCP Server**

bash

cd graphiti/mcp\_server

uv run graphiti\_mcp\_server.py --transport sse

**2. Claude Desktop配置**

json

{

"mcpServers": {

"notion-memory": {

"command": "uv",

"args": [

"run",

"--directory", "/path/to/graphiti/mcp\_server",

"graphiti\_mcp\_server.py",

"--transport", "stdio"

],

"env": {

"NOTION\_API\_KEY": "your\_notion\_key",

"OPENAI\_API\_KEY": "your\_openai\_key",

"NEO4J\_URI": "bolt://localhost:7687",

"NEO4J\_USER": "neo4j",

"NEO4J\_PASSWORD": "password"

}

}

}

}

**3. 使用示例**

在Claude中：

1. 首先同步Notion：sync\_notion

2. 然后搜索：search\_notion\_memory "项目管理相关的页面"

**核心优势**

✅ **基于Graphiti内置MCP** - 无需额外框架  
✅ **官方Notion SDK** - 稳定可靠的API调用  
✅ **图谱即索引** - 只存储关系，内容实时从Notion获取  
✅ **五种关系模型** - 完整捕捉Notion结构  
✅ **Embedding搜索** - AI驱动的语义检索  
✅ **即插即用** - 直接集成到现有AI工作流

这样就能充分利用Graphiti的MCP基础设施，同时用官方Notion SDK确保API调用的稳定性！