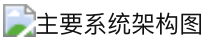
Supermemory 技术架构设计

1. 架构设计

1.1 整体架构概述

Supermemory 采用先进的**双引擎架构**，将**时序记忆引擎**与**RAG增强引擎**深度融合，构建企业级智能记忆系统。系统通过智能融合检索技术，实现时序记忆与外部知识库的无缝整合，为用户提供更加智能和个性化的交互体验。

RAG（Retrieval-Augmented Generation）技术作为Supermemory的核心增强模块，不是独立存在，而是深度集成到记忆检索流程中，形成**智能融合检索引擎**，大幅提升回答的准确性和全面性。

 主要系统架构图

2. 技术描述

2.1 核心技术栈

Supermemory采用先进的**双引擎架构**，将**时序记忆引擎**与**RAG增强引擎**深度融合：

- **前端**: React@18 + TypeScript + Tailwind CSS + Vite
- **后端**: Node.js + Express@4 + TypeScript
- **数据库**: PostgreSQL + Redis + Pinecone (向量数据库) + Zep Graphiti (时序知识图谱)
- **嵌入服务**: OpenAI text-embedding-3-small + sentence-transformers
- **消息队列**: Redis + Bull Queue
- **缓存系统**: Redis (多层缓存策略)
- **监控**: Prometheus + Grafana
- **部署**: Docker + Kubernetes

2.1.1 双引擎架构

时序记忆引擎: 基于Zep Graphiti时序知识图谱，提供卓越的记忆连贯性和时序推理能力

RAG增强引擎: 深度集成的外部知识检索，支持多源知识融合，显著提升回答准确性

融合检索引擎: 智能融合时序记忆与外部知识，实现1+1>2的增强效果

2.2 Zep时序记忆架构

Supermemory采用先进的**Zep时序记忆架构**，基于Graphiti时序知识图谱技术，构建企业级AI记忆系统：

2.2.1 Graphiti时序知识图谱核心

- **时序实体管理**
  - 自动提取和跟踪实体的时间演化

    - 支持实体状态变化的历史记录
    - 动态实体关系建模和更新

- **动态关系推理**
    - 基于时间序列的关系强度计算
    - 支持因果关系和时序依赖分析
    - 自适应关系权重调整机制

- **跨会话记忆合成**
    - 智能合并多会话信息
    - 保持长期上下文连贯性
    - 支持复杂信息综合和推理

2.2.2 企业级优化特性

- **高性能检索引擎**: 基于时序图的快速检索，支持复杂查询
- **智能去重机制**: 自动识别和合并重复信息，减少冗余
- **动态知识整合**: 实时整合新信息到现有知识图谱
- **时序感知推理**: 考虑时间因素的智能推理和预测
- **多源数据融合**: 支持多种数据源的统一建模
- **自适应学习**: 基于用户反馈的持续优化

2.2.3 Zep架构优势

- **卓越性能**: 在DMR和LongMemEval基准测试中超越MemGPT
- **企业就绪**: 内置合规性和安全性保障
- **时序感知**: 原生支持时间维度的记忆管理
- **动态整合**: 实时整合多源知识，减少冗余检索
- **成本优化**: 降低Token使用量和延迟
- **可扩展性**: 支持大规模企业级部署
- **可解释性**: 提供清晰的推理路径和决策依据

2.3 RAG增强引擎架构

RAG增强引擎是Supermemory的核心组件，深度集成到系统的检索流程中：

2.3.1 融合检索引擎核心

- **智能融合算法**
  - 基于语义相似度和时序关联的混合检索策略
  - 动态权重调整（记忆权重0.6-0.8，RAG权重0.2-0.4）
  - 时间衰减因子和相关性提升机制
- **并行检索优化**
  - 同时搜索时序记忆库和外部知识库
  - 异步处理机制，确保响应速度
  - 智能缓存策略，提升检索效率
- **质量评估体系**
  - 多维度相关性评分
  - 时效性评估和置信度计算
  - 动态上下文长度优化

2.3.2 多源知识适配器

- **支持的知识源类型**
  - Elasticsearch: 企业文档和日志检索
  - Chroma DB: 向量化知识库存储
  - Weaviate: 语义知识图谱查询
  - 本地文件系统: Markdown、PDF、TXT等格式
  - 外部API接口: 第三方知识服务集成
- **统一接入层**
  - 标准化的知识源配置接口
  - 动态连接池管理和健康检查
  - 智能重试机制和故障转移

2.3.3 知识源管理器

- **生命周期管理**
  - 知识源的注册、更新、删除操作
  - 实时同步和增量更新机制
  - 版本控制和变更追踪
- **优先级和权重系统**
  - 可配置的知识源优先级（1-10）
  - 动态权重调整算法
  - 基于历史效果的智能优化

2.3.4 企业级特性

- **高性能检索**: 基于HNSW算法的近似最近邻搜索
- **智能缓存**: 多层缓存策略，热点查询结果缓存
- **监控告警**: 实时性能监控和质量告警
- **安全合规**: 数据加密传输和访问权限控制

- **可扩展性**: 水平扩展支持大规模知识库

3. 路由定义

路由	用途
/dashboard	管理控制台主页，显示项目概览和使用统计
/projects	项目管理页面，创建和配置AI项目
/proxy-config	代理配置页面，设置LLM模型和记忆策略
/memory	记忆管理页面，查看和管理对话历史
/segmentation-config	智能分段配置页面，设置分段策略和检索参数
/token-management	Token管理页面，监控使用量和优化策略配置
/analytics	分析仪表板，性能和成本分析
/settings	系统设置页面，账户和安全配置
/api-docs	API文档页面，开发者集成指南

4. API定义

4.1 核心代理API

透明代理接口

```
POST /v1/chat/completions
```

请求参数:

参数名	参数类型	是否必需	描述
model	string	true	LLM模型名称 (gpt-4, claude-3, etc.)
messages	array	true	对话消息数组
temperature	number	false	生成温度 (0-2)
max_tokens	number	false	最大token数
stream	boolean	false	是否流式响应
memory_config	object	false	记忆配置参数

响应:

参数名	参数类型	描述
id	string	请求唯一标识

object	string	响应对象类型
created	number	创建时间戳
model	string	使用的模型
choices	array	生成的回复选项
usage	object	Token使用统计
memory_info	object	记忆处理信息

示例请求:

```
{
  "model": "gpt-4",
  "messages": [
    {"role": "user", "content": "你好，我想了解AI的发展历史"}
  ],
  "temperature": 0.7,
  "memory_config": {
      "enable_memory": true,
      "temporal_graph": {
        "enabled": true,
        "time_window": "7d",
        "entity_tracking": true,
        "relationship_inference": true
      },
      "cross_session": {
        "enabled": true,
        "synthesis_depth": 10,
        "coherence_threshold": 0.8
      },
      "optimization": {
        "enable_deduplication": true,
        "enable_temporal_reasoning": true,
        "performance_mode": "balanced"
      }
    }
}
```

示例响应:

```
{
  "id": "chatcmpl-abc123",
  "object": "chat.completion",
  "created": 1677652288,
```

```
  "model": "gpt-4",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "AI的发展历史可以追溯到1950年代..."
      },
      "finish_reason": "stop"
    }
  ],
  "usage": {
    "prompt_tokens": 150,
    "completion_tokens": 200,
    "total_tokens": 350,
    "memory_tokens": 50
  },
  "memory_info": {
    "temporal_graph_updated": true,
    "entities_tracked": 7,
    "cross_session_memories_synthesized": 4,
    "temporal_relationships_inferred": 12,
    "duplicates_removed": 2,
    "performance_metrics": {
      "retrieval_time_ms": 32,
      "synthesis_time_ms": 85,
      "token_reduction": "34%",
      "latency_improvement": "28%"
    }
  }
}
```

Zep时序记忆管理API

```
# Graphiti时序知识图谱 API
GET /v1/memory/temporal-graph          # 获取时序知识图谱
POST /v1/memory/temporal-graph/build   # 构建时序图谱
PUT /v1/memory/temporal-graph/update   # 更新图谱结构
GET /v1/memory/temporal-graph/query    # 时序图谱查询

# 时序实体管理 API
GET /v1/memory/entities                # 获取实体列表
POST /v1/memory/entities/extract       # 提取新实体
PUT /v1/memory/entities/track          # 跟踪实体变化
GET /v1/memory/entities/timeline       # 获取实体时间线
POST /v1/memory/entities/merge         # 合并重复实体
```

```
# 动态关系推理 API
GET /v1/memory/relationships          # 获取关系网络
POST /v1/memory/relationships/infer   # 推理新关系
PUT /v1/memory/relationships/weight   # 更新关系权重
GET /v1/memory/relationships/temporal # 获取时序关系
POST /v1/memory/relationships/causal  # 因果关系分析

# 跨会话记忆合成 API
GET /v1/memory/cross-session          # 获取跨会话记忆
POST /v1/memory/cross-session/synthesize # 合成会话信息
PUT /v1/memory/cross-session/coherence   # 维护连贯性
GET /v1/memory/cross-session/context     # 获取长期上下文

# 传统记忆管理 API（兼容性）
GET /v1/memory/conversations          # 获取对话历史
POST /v1/memory/clear                 # 清理记忆
PUT /v1/memory/update                 # 更新记忆
GET /v1/memory/search                 # 搜索记忆

# 智能去重与优化 API
POST /v1/memory/deduplication         # 执行去重操作
GET /v1/memory/deduplication/status   # 获取去重状态
PUT /v1/memory/deduplication/config   # 配置去重策略
GET /v1/memory/optimization/insights  # 获取优化洞察
```

智能分段配置API

```
GET /v1/segmentation/config
PUT /v1/segmentation/config
POST /v1/segmentation/test
GET /v1/segmentation/performance
```

性能优化与成本控制API

```
# Token管理 API
GET /v1/token/usage                   # 获取Token使用情况
GET /v1/token/statistics              # 获取使用统计
PUT /v1/token/config                  # 配置Token管理
POST /v1/token/optimize               # 优化Token使用

# 性能优化 API
GET /v1/performance/metrics           # 获取性能指标
POST /v1/performance/optimize         # 触发性能优化
PUT /v1/performance/parameters        # 调整优化参数
GET /v1/performance/recommendations   # 获取优化建议
```

```
# 成本控制 API
GET /v1/cost/analysis                    # 成本分析
POST /v1/cost/optimize                   # 成本优化
PUT /v1/cost/budget                      # 设置预算限制
GET /v1/cost/forecast                    # 成本预测

# 缓存管理 API
GET /v1/cache/status                     # 缓存状态
POST /v1/cache/clear                     # 清理缓存
PUT /v1/cache/config                     # 配置缓存策略
GET /v1/cache/analytics                  # 缓存分析

# 自动调优 API
POST /v1/autotune/start                  # 启动自动调优
GET /v1/autotune/status                  # 调优状态
PUT /v1/autotune/config                  # 调优配置
GET /v1/autotune/results                 # 调优结果
```

项目管理API

```
GET /v1/projects
POST /v1/projects
PUT /v1/projects/{id}
DELETE /v1/projects/{id}
```

4.2 Zep时序记忆管理API详细示例

4.2.1 时序实体管理API示例

获取实体时间线

```
GET /v1/memory/entities/timeline?entity_id=user_zhang_san
Authorization: Bearer your-api-key
```

响应示例:

```
{
  "entity": {
    "id": "user_zhang_san",
    "type": "person",
    "name": "张三",
    "current_state": {
      "role": "高级产品经理",
      "company": "科技公司",
      "skills": ["AI", "产品设计", "用户体验", "机器学习"]
    }
  },
```

```
  "timeline": [
    {
      "timestamp": "2024-01-15T10:30:00Z",
      "event_type": "role_change",
      "description": "晋升为高级产品经理",
      "confidence": 0.95
    },
    {
      "timestamp": "2024-01-14T15:20:00Z",
      "event_type": "skill_acquisition",
      "description": "学习机器学习技能",
      "confidence": 0.88
    }
  ],
  "relationships": [
    {
      "target_entity": "ai_project",
      "relation_type": "manages",
      "strength": 0.9,
      "temporal_pattern": "increasing"
    }
  ]
}
```

跟踪实体变化

```
PUT /v1/memory/entities/track
Content-Type: application/json
Authorization: Bearer your-api-key

{
  "entity_id": "user_zhang_san",
  "changes": {
    "role": "高级产品经理",
    "new_skills": ["数据分析"],
    "project_involvement": "AI产品开发"
  },
  "timestamp": "2024-01-15T11:00:00Z",
  "confidence": 0.92
}
```

4.2.2 动态关系推理API示例

推理新关系

```
POST /v1/memory/relationships/infer
Content-Type: application/json
Authorization: Bearer your-api-key

{
  "context": {
    "conversation_id": "conv_123",
    "new_message": "我们讨论的AI项目预算是多少？"
  },
  "inference_params": {
    "temporal_window": "7d",
    "confidence_threshold": 0.7,
    "max_relationships": 10
  }
}
```

响应示例:

```
{
  "inferred_relationships": [
    {
      "source_entity": "user_zhang_san",
      "target_entity": "ai_project_budget",
      "relation_type": "inquires_about",
      "confidence": 0.92,
      "temporal_context": "2024-01-15T09:15:00Z",
      "supporting_evidence": [
        "直接询问预算信息",
        "历史上多次关注成本问题"
      ]
    },
    {
      "source_entity": "ai_project",
      "target_entity": "budget_constraint",
      "relation_type": "has_constraint",
      "confidence": 0.85,
      "temporal_context": "2024-01-15T09:20:00Z",
      "causal_chain": ["项目需求", "资源限制", "预算约束"]
    }
  ],
  "temporal_patterns": {
    "budget_discussions": {
      "frequency": "weekly",
      "trend": "increasing",
      "peak_times": ["Monday morning", "Friday afternoon"]
```

```
    }
  }
}
```

4.2.3 跨会话记忆合成API示例

合成会话信息

```
POST /v1/memory/cross-session/synthesize
Content-Type: application/json
Authorization: Bearer your-api-key

{
  "query": "AI项目的技术选型讨论",
  "synthesis_params": {
    "session_count": 5,
    "time_window": "30d",
    "coherence_threshold": 0.8,
    "include_temporal_evolution": true
  }
}
```

响应示例:

```
{
  "synthesized_memory": {
    "topic": "AI项目技术选型",
    "evolution_timeline": [
      {
        "phase": "初期探索",
        "timeframe": "2024-01-01 to 2024-01-07",
        "key_decisions": [
          "考虑多种架构方案",
          "评估Transformer vs CNN"
        ],
        "confidence": 0.89
      },
      {
        "phase": "方案确定",
        "timeframe": "2024-01-08 to 2024-01-15",
        "key_decisions": [
          "选择Transformer架构",
          "确定使用预训练模型"
        ],
        "confidence": 0.94
      }
```

```
    ],
    "cross_session_insights": {
      "consistent_preferences": [
        "偏好开源解决方案",
        "重视性能和成本平衡"
      ],
      "evolving_requirements": [
        "从通用性转向专业化",
        "增加对实时性的要求"
      ]
    },
    "temporal_relationships": [
      {
        "entity1": "Transformer",
        "entity2": "performance_requirement",
        "relation_evolution": "strengthening",
        "confidence_trend": "increasing"
      }
    ]
  }
}
```

4.2.4 性能优化API示例

获取性能指标

```
GET /v1/performance/metrics
Authorization: Bearer your-api-key
```

响应示例:

```
{
  "current_metrics": {
    "avg_response_time_ms": 150,
    "memory_usage_percent": 65,
    "cache_hit_rate": 0.85,
    "cost_per_query": 0.032,
    "throughput_qps": 120
  },
  "optimization_status": {
    "auto_tuning_enabled": true,
    "last_optimization": "2024-01-15T08:00:00Z",
    "next_optimization": "2024-01-15T20:00:00Z"
  },
  "recommendations": [
    {
```

```
      "type": "cache_optimization",
      "description": "增加缓存大小可提升15%性能",
      "impact": "medium",
      "effort": "low"
    }
  ]
}
```

4.3 RAG知识库融合API

4.3.1 知识源管理API

获取所有知识源

```
GET /v1/rag-knowledge/sources
Authorization: Bearer your-api-key
```

响应示例:

```
{
  "success": true,
  "data": [
    {
      "id": "rag-001",
      "name": "企业文档库",
      "type": "elasticsearch",
      "status": "connected",
      "config": {
        "url": "https://es.example.com:9200",
        "index": "company_docs",
        "apiKey": "***"
      },
      "documentCount": 15420,
      "lastSync": "2024-01-15T10:30:00Z",
      "createdAt": "2024-01-01T00:00:00Z"
    },
    {
      "id": "rag-002",
      "name": "技术知识库",
      "type": "chroma",
      "status": "connected",
      "config": {
        "url": "http://chroma.example.com:8000",
        "collection": "tech_knowledge"
      },
      "documentCount": 8750,
      "lastSync": "2024-01-15T09:45:00Z",
```

```
      "createdAt": "2024-01-05T00:00:00Z"
    }
  ]
}
```

添加知识源

```
POST /v1/rag-knowledge/sources
Authorization: Bearer your-api-key
Content-Type: application/json
```

请求参数:

参数名	参数类型	是否必需	描述
name	string	true	知识源名称
type	string	true	知识源类型 (elasticsearch/chroma/weaviate/local_files/database/api)
config	object	true	知识源配置信息

请求示例:

```
{
  "name": "新文档库",
  "type": "elasticsearch",
  "config": {
    "url": "https://new-es.example.com:9200",
    "index": "new_docs",
    "apiKey": "your-api-key"
  }
}
```

更新知识源

```
PUT /v1/rag-knowledge/sources/{sourceId}
Authorization: Bearer your-api-key
Content-Type: application/json
```

删除知识源

```
DELETE /v1/rag-knowledge/sources/{sourceId}
Authorization: Bearer your-api-key
```

测试知识源连接

```
POST /v1/rag-knowledge/sources/{sourceId}/test
Authorization: Bearer your-api-key
```

响应示例:

```
{
  "success": true,
  "connected": true,
  "latency_ms": 45,
  "document_count": 15420,
  "last_test": "2024-01-15T11:00:00Z"
}
```

4.3.2 融合搜索API

执行融合搜索

```
POST /v1/rag-knowledge/search
Authorization: Bearer your-api-key
Content-Type: application/json
```

请求参数:

参数名	参数类型	是否必需	描述
query	string	true	搜索查询
sources	array	false	指定搜索的知识源ID列表
limit	number	false	返回结果数量限制 (默认10)
threshold	number	false	相似度阈值 (默认0.7)
includeMetadata	boolean	false	是否包含元数据 (默认false)
timeRange	object	false	时间范围过滤

请求示例:

```
{
  "query": "如何优化数据库性能",
  "sources": ["rag-001", "rag-002"],
  "limit": 15,
  "threshold": 0.75,
  "includeMetadata": true,
  "timeRange": {
```

```
    "start": "2024-01-01T00:00:00Z",
    "end": "2024-01-15T23:59:59Z"
  }
}
```

响应示例:

```
{
  "success": true,
  "data": {
    "supermemoryResults": [
      {
        "id": "mem-001",
        "content": "之前讨论过数据库索引优化...",
        "score": 0.92,
        "source": "supermemory",
        "timestamp": "2024-01-10T14:30:00Z",
        "fusionScore": 0.95
      }
    ],
    "ragResults": [
      {
        "id": "doc-001",
        "content": "数据库性能优化最佳实践包括...",
        "score": 0.88,
        "source": "rag-001",
        "metadata": {
          "title": "数据库优化指南",
          "author": "技术团队",
          "category": "数据库"
        },
        "timestamp": "2024-01-05T10:00:00Z",
        "fusionScore": 0.89
      }
    ],
    "fusedResults": [
      {
        "id": "mem-001",
        "content": "之前讨论过数据库索引优化...",
        "score": 0.92,
        "source": "supermemory",
        "fusionScore": 0.95,
        "timestamp": "2024-01-10T14:30:00Z"
      },
      {
        "id": "doc-001",
```

```
        "content": "数据库性能优化最佳实践包括...",
        "score": 0.88,
        "source": "rag-001",
        "fusionScore": 0.89,
        "timestamp": "2024-01-05T10:00:00Z"
      }
    ],
    "searchMetrics": {
      "totalResults": 2,
      "searchTime_ms": 125,
      "sourcesSearched": ["supermemory", "rag-001", "rag-002"],
      "cacheHit": false
    }
  }
}
```

4.3.3 增强聊天API

增强聊天完成 (集成RAG)

```
POST /v1/chat/completions
Authorization: Bearer your-api-key
Content-Type: application/json
```

扩展的请求参数:

参数名	参数类型	是否必需	描述
model	string	true	LLM模型名称
messages	array	true	对话消息数组
useRAG	boolean	false	是否启用RAG融合 (默认true)
ragSources	array	false	指定使用的RAG知识源
ragConfig	object	false	RAG配置参数

请求示例:

```
{
  "model": "gpt-4",
  "messages": [
    {"role": "user", "content": "如何提升系统性能? "}
  ],
  "useRAG": true,
  "ragSources": ["rag-001", "rag-002"],
  "ragConfig": {
```

```
    "threshold": 0.75,
    "maxResults": 10,
    "includeMetadata": true
  },
  "memory_config": {
    "enable_memory": true,
    "temporal_graph": {
      "enabled": true,
      "time_window": "7d"
    }
  }
}
```

响应示例:

```
{
  "id": "chatcmpl-rag123",
  "object": "chat.completion",
  "created": 1705312800,
  "model": "gpt-4",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "基于您的历史讨论和企业知识库，系统性能提升可以从以下几个方面入
手..."
      },
      "finish_reason": "stop"
    }
  ],
  "usage": {
    "prompt_tokens": 280,
    "completion_tokens": 150,
    "total_tokens": 430,
    "memory_tokens": 80,
    "rag_tokens": 50
  },
  "memory_info": {
    "temporal_graph_updated": true,
    "entities_tracked": 5,
    "relationships_inferred": 3
  },
  "rag_info": {
    "sources_searched": ["supermemory", "rag-001", "rag-002"],
    "results_found": 8,
```

```
    "fusion_applied": true,
    "search_time_ms": 125
  }
}
```

测试RAG功能

```
POST /v1/chat/test-rag
Authorization: Bearer your-api-key
Content-Type: application/json
```

请求示例:

```
{
  "query": "测试查询",
  "sources": ["rag-001"]
}
```

响应示例:

```
{
  "success": true,
  "data": {
    "query": "测试查询",
    "results": [
      {
        "id": "test-001",
        "content": "测试内容...",
        "score": 0.85,
        "source": "rag-001"
      }
    ],
    "performance": {
      "search_time_ms": 45,
      "total_results": 1
    }
  }
}
```

5. 服务器架构图

```
graph TD
    A[负载均衡器] --> B[API网关集群]
    B --> C[认证服务]
```

```
    B --> D[代理服务集群]

    D --> E[记忆检索服务]
    D --> F[上下文处理服务]
    D --> G[LLM调用服务]

    E --> H[向量搜索集群]
    F --> I[缓存集群]
    G --> J[外部LLM APIs]

    D --> K[异步任务队列]
    K --> L[记忆更新服务]
    L --> M[数据库集群]

    subgraph "入口层"
        A
        B
    end

    subgraph "服务层"
        C
        D
        E
        F
        G
        L
    end

    subgraph "数据层"
        H
        I
        M
    end

    subgraph "队列层"
        K
    end
```

6. 数据模型

6.1 数据模型定义

6.1.1 核心实体关系图

```
erDiagram
    USER ||--o{ PROJECT : owns
```

```
PROJECT ||--o{ CONVERSATION : contains
CONVERSATION ||--o{ MESSAGE : includes
MESSAGE ||--o{ MEMORY_CHUNK : generates
PROJECT ||--o{ API_KEY : has
PROJECT ||--o{ SEGMENTATION_CONFIG : configures
PROJECT ||--o{ TOKEN_CONFIG : configures
USER ||--o{ USAGE_LOG : generates
USER ||--o{ TOKEN_USAGE_LOG : generates

MEMORY_CHUNK ||--o{ TEMPORAL_ENTITY : "extracted from"
TEMPORAL_ENTITY ||--o{ TEMPORAL_RELATION : "connected by"
TEMPORAL_ENTITY }o--o{ GRAPHITI_GRAPH : "part of"
GRAPHITI_GRAPH ||--o{ TEMPORAL_EVOLUTION : "evolves through"
TEMPORAL_ENTITY ||--o{ ENTITY_TIMELINE : "has timeline"
TEMPORAL_RELATION ||--o{ RELATION_TIMELINE : "has timeline"

PROJECT ||--o{ RAG_KNOWLEDGE_SOURCE : configures
RAG_KNOWLEDGE_SOURCE ||--o{ RAG_SEARCH_LOG : generates
RAG_KNOWLEDGE_SOURCE ||--o{ RAG_SYNC_LOG : tracks
MESSAGE ||--o{ RAG_SEARCH_RESULT : enhances
RAG_SEARCH_RESULT }o--|| RAG_KNOWLEDGE_SOURCE : "sourced from"

USER {
    uuid id PK
    string email
    string password_hash
    string name
    string plan
    timestamp created_at
    timestamp updated_at
}

PROJECT {
    uuid id PK
    uuid user_id FK
    string name
    string description
    json config
    string status
    timestamp created_at
    timestamp updated_at
}

CONVERSATION {
    uuid id PK
    uuid project_id FK
```

```
    string session_id
    json metadata
    timestamp created_at
    timestamp updated_at
}

MESSAGE {
    uuid id PK
    uuid conversation_id FK
    string role
    text content
    json metadata
    integer token_count
    timestamp created_at
}

MEMORY_CHUNK {
    uuid id PK
    uuid message_id FK
    text content
    vector embedding
    float relevance_score
    string chunk_type
    integer chunk_index
    json metadata
    timestamp created_at
}

TEMPORAL_ENTITY {
    uuid id PK
    uuid memory_chunk_id FK
    string entity_type
    string entity_name
    json entity_attributes
    vector entity_embedding
    float importance_score
    timestamp first_seen
    timestamp last_updated
    json temporal_context
    string entity_status
    timestamp created_at
    timestamp updated_at
}

TEMPORAL_RELATION {
    uuid id PK
```

```
    uuid source_entity_id FK
    uuid target_entity_id FK
    string relation_type
    float relation_strength
    json relation_metadata
    timestamp relation_start
    timestamp relation_end
    string relation_status
    float confidence_score
    timestamp created_at
    timestamp updated_at
}

GRAPHITI_GRAPH {
    uuid id PK
    uuid project_id FK
    string graph_name
    json temporal_structure
    integer entity_count
    integer relation_count
    timestamp temporal_window_start
    timestamp temporal_window_end
    json synthesis_metadata
    timestamp last_evolved
    timestamp created_at
    timestamp updated_at
}

TEMPORAL_EVOLUTION {
    uuid id PK
    uuid graph_id FK
    string evolution_type
    json before_state
    json after_state
    string evolution_reason
    timestamp evolution_time
    float confidence_score
    timestamp created_at
}

ENTITY_TIMELINE {
    uuid id PK
    uuid entity_id FK
    json timeline_events
    timestamp timeline_start
    timestamp timeline_end
```

```
    json evolution_pattern
    timestamp created_at
    timestamp updated_at
}

RELATION_TIMELINE {
    uuid id PK
    uuid relation_id FK
    json timeline_events
    timestamp timeline_start
    timestamp timeline_end
    json strength_evolution
    timestamp created_at
    timestamp updated_at
}

SEGMENTATION_CONFIG {
    uuid id PK
    uuid project_id FK
    integer max_chunk_size
    integer overlap_size
    string segmentation_strategy
    json embedding_config
    float similarity_threshold
    json weights_config
    timestamp created_at
    timestamp updated_at
}

TOKEN_CONFIG {
    uuid id PK
    uuid project_id FK
    integer max_context_tokens
    integer reserved_tokens
    json compression_config
    json priority_weights
    boolean auto_optimization
    timestamp created_at
    timestamp updated_at
}

TOKEN_USAGE_LOG {
    uuid id PK
    uuid user_id FK
    uuid project_id FK
    uuid conversation_id FK
```

```
    integer input_tokens
    integer output_tokens
    integer memory_tokens
    integer compressed_tokens
    float compression_ratio
    string optimization_strategy
    timestamp created_at
}

API_KEY {
    uuid id PK
    uuid project_id FK
    string key_hash
    string name
    json permissions
    timestamp expires_at
    timestamp created_at
}

USAGE_LOG {
    uuid id PK
    uuid user_id FK
    uuid project_id FK
    string endpoint
    integer tokens_used
    float cost
    timestamp created_at
}

RAG_KNOWLEDGE_SOURCE {
    uuid id PK
    uuid project_id FK
    string name
    string source_type
    json config
    string status
    integer document_count
    timestamp last_sync
    json sync_config
    json search_config
    timestamp created_at
    timestamp updated_at
}

RAG_SEARCH_LOG {
    uuid id PK
```

```
        uuid source_id FK
        uuid message_id FK
        string query
        json search_params
        integer results_count
        float search_time_ms
        json performance_metrics
        timestamp created_at
    }

    RAG_SYNC_LOG {
        uuid id PK
        uuid source_id FK
        string sync_type
        string sync_status
        integer documents_processed
        integer documents_added
        integer documents_updated
        integer documents_deleted
        json sync_metadata
        string error_message
        timestamp sync_start
        timestamp sync_end
        timestamp created_at
    }

    RAG_SEARCH_RESULT {
        uuid id PK
        uuid message_id FK
        uuid source_id FK
        string result_id
        text content
        float score
        float fusion_score
        json metadata
        string result_type
        timestamp result_timestamp
        timestamp created_at
    }
```

6.2 数据定义语言

用户表 (users)

```
-- 创建用户表
CREATE TABLE users (
```

```
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    name VARCHAR(100) NOT NULL,
    plan VARCHAR(20) DEFAULT 'free' CHECK (plan IN ('free', 'pro',
'enterprise')),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- 创建索引
CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_users_plan ON users(plan);
```

项目表 (projects)

```
-- 创建项目表
CREATE TABLE projects (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    name VARCHAR(100) NOT NULL,
    description TEXT,
    config JSONB DEFAULT '{}',
    status VARCHAR(20) DEFAULT 'active' CHECK (status IN ('active',
'paused', 'deleted')),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- 创建索引
CREATE INDEX idx_projects_user_id ON projects(user_id);
CREATE INDEX idx_projects_status ON projects(status);
```

对话表 (conversations)

```
-- 创建对话表
CREATE TABLE conversations (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    project_id UUID NOT NULL REFERENCES projects(id) ON DELETE CASCADE,
    session_id VARCHAR(255) NOT NULL,
    metadata JSONB DEFAULT '{}',
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- 创建索引
```

```
CREATE INDEX idx_conversations_project_id ON conversations(project_id);
CREATE INDEX idx_conversations_session_id ON conversations(session_id);
CREATE INDEX idx_conversations_created_at ON conversations(created_at
DESC);
```

消息表 (messages)

```
-- 创建消息表
CREATE TABLE messages (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    conversation_id UUID NOT NULL REFERENCES conversations(id) ON DELETE
CASCADE,
    role VARCHAR(20) NOT NULL CHECK (role IN ('user', 'assistant',
'system')),
    content TEXT NOT NULL,
    metadata JSONB DEFAULT '{}',
    token_count INTEGER DEFAULT 0,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- 创建索引
CREATE INDEX idx_messages_conversation_id ON messages(conversation_id);
CREATE INDEX idx_messages_created_at ON messages(created_at DESC);
CREATE INDEX idx_messages_token_count ON messages(token_count);
```

记忆块表 (memory_chunks)

```
-- 创建记忆块表
CREATE TABLE memory_chunks (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    message_id UUID NOT NULL REFERENCES messages(id) ON DELETE CASCADE,
    content TEXT NOT NULL,
    embedding VECTOR(1536), -- 使用pgvector扩展
    relevance_score FLOAT DEFAULT 0.0,
    chunk_type VARCHAR(50) DEFAULT 'semantic' CHECK (chunk_type IN
('semantic', 'paragraph', 'fixed', 'sliding')),
    chunk_index INTEGER DEFAULT 0,
    metadata JSONB DEFAULT '{}',
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- 创建向量索引
CREATE INDEX idx_memory_chunks_embedding ON memory_chunks USING ivfflat
(embedding vector_cosine_ops);
CREATE INDEX idx_memory_chunks_message_id ON memory_chunks(message_id);
CREATE INDEX idx_memory_chunks_relevance_score ON
```

```
memory_chunks(relevance_score DESC);
CREATE INDEX idx_memory_chunks_chunk_type ON memory_chunks(chunk_type);
CREATE INDEX idx_memory_chunks_chunk_index ON memory_chunks(chunk_index);
```

分段配置表 (segmentation_configs)

```
-- 创建分段配置表
CREATE TABLE segmentation_configs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    project_id UUID NOT NULL REFERENCES projects(id) ON DELETE CASCADE,
    max_chunk_size INTEGER DEFAULT 512,
    overlap_size INTEGER DEFAULT 50,
    segmentation_strategy VARCHAR(50) DEFAULT 'semantic' CHECK
(segmentation_strategy IN ('semantic', 'paragraph', 'fixed', 'sliding',
'hybrid')),
    embedding_config JSONB DEFAULT '{"model": "text-embedding-3-small",
"dimensions": 1536}',
    similarity_threshold FLOAT DEFAULT 0.7,
    weights_config JSONB DEFAULT '{"semantic": 0.4, "temporal": 0.3,
"importance": 0.2, "position": 0.1}',
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- 创建索引
CREATE INDEX idx_segmentation_configs_project_id ON
segmentation_configs(project_id);
CREATE UNIQUE INDEX idx_segmentation_configs_project_unique ON
segmentation_configs(project_id);
```

Token配置表 (token_configs)

```
-- 创建Token配置表
CREATE TABLE token_configs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    project_id UUID NOT NULL REFERENCES projects(id) ON DELETE CASCADE,
    max_context_tokens INTEGER DEFAULT 4096,
    reserved_tokens INTEGER DEFAULT 1000,
    compression_config JSONB DEFAULT '{"enable_compression": true,
"compression_ratio": 0.7, "summary_model": "gpt-3.5-turbo"}',
    priority_weights JSONB DEFAULT '{"temporal": 0.3, "relevance": 0.4,
"importance": 0.2, "user_preference": 0.1}',
    auto_optimization BOOLEAN DEFAULT true,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

```sql
-- 创建索引
CREATE INDEX idx_token_configs_project_id ON token_configs(project_id);
CREATE UNIQUE INDEX idx_token_configs_project_unique ON
token_configs(project_id);
```

Token使用日志表 (token_usage_logs)

```sql
-- 创建Token使用日志表
CREATE TABLE token_usage_logs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    project_id UUID REFERENCES projects(id) ON DELETE SET NULL,
    conversation_id UUID REFERENCES conversations(id) ON DELETE SET NULL,
    input_tokens INTEGER DEFAULT 0,
    output_tokens INTEGER DEFAULT 0,
    memory_tokens INTEGER DEFAULT 0,
    compressed_tokens INTEGER DEFAULT 0,
    compression_ratio FLOAT DEFAULT 1.0,
    optimization_strategy VARCHAR(100),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- 创建索引
CREATE INDEX idx_token_usage_logs_user_id ON token_usage_logs(user_id);
CREATE INDEX idx_token_usage_logs_project_id ON
token_usage_logs(project_id);
CREATE INDEX idx_token_usage_logs_conversation_id ON
token_usage_logs(conversation_id);
CREATE INDEX idx_token_usage_logs_created_at ON token_usage_logs(created_at
DESC);

-- 创建分区表（按月分区）
CREATE TABLE token_usage_logs_y2024m01 PARTITION OF token_usage_logs
FOR VALUES FROM ('2024-01-01') TO ('2024-02-01');
```

API密钥表 (api_keys)

```sql
-- 创建API密钥表
CREATE TABLE api_keys (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    project_id UUID NOT NULL REFERENCES projects(id) ON DELETE CASCADE,
    key_hash VARCHAR(255) NOT NULL UNIQUE,
    name VARCHAR(100) NOT NULL,
    permissions JSONB DEFAULT '{}',
    expires_at TIMESTAMP WITH TIME ZONE,
```

```
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- 创建索引
CREATE INDEX idx_api_keys_project_id ON api_keys(project_id);
CREATE INDEX idx_api_keys_key_hash ON api_keys(key_hash);
```

使用日志表 (usage_logs)

```
-- 创建使用日志表
CREATE TABLE usage_logs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    project_id UUID REFERENCES projects(id) ON DELETE SET NULL,
    endpoint VARCHAR(255) NOT NULL,
    tokens_used INTEGER DEFAULT 0,
    cost FLOAT DEFAULT 0.0,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- 创建索引
CREATE INDEX idx_usage_logs_user_id ON usage_logs(user_id);
CREATE INDEX idx_usage_logs_project_id ON usage_logs(project_id);
CREATE INDEX idx_usage_logs_endpoint ON usage_logs(endpoint);
CREATE INDEX idx_usage_logs_created_at ON usage_logs(created_at DESC);
```

6.2.2 Mem0记忆架构数据表

记忆实体表 (memory_entities)

```
-- 创建记忆实体表
CREATE TABLE memory_entities (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    memory_chunk_id UUID NOT NULL REFERENCES memory_chunks(id) ON DELETE
CASCADE,
    entity_type VARCHAR(50) NOT NULL CHECK (entity_type IN ('person',
'organization', 'location', 'concept', 'event', 'topic')),
    entity_name VARCHAR(255) NOT NULL,
    entity_attributes JSONB DEFAULT '{}',
    entity_embedding VECTOR(1536),
    importance_score FLOAT DEFAULT 0.5,
    confidence_score FLOAT DEFAULT 0.8,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

```
-- 创建索引
CREATE INDEX idx_memory_entities_chunk_id ON
memory_entities(memory_chunk_id);
CREATE INDEX idx_memory_entities_type ON memory_entities(entity_type);
CREATE INDEX idx_memory_entities_name ON memory_entities(entity_name);
CREATE INDEX idx_memory_entities_embedding ON memory_entities USING ivfflat
(entity_embedding vector_cosine_ops);
CREATE INDEX idx_memory_entities_importance ON
memory_entities(importance_score DESC);
```

记忆关系表 (memory_relations)

```
-- 创建记忆关系表
CREATE TABLE memory_relations (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    source_entity_id UUID NOT NULL REFERENCES memory_entities(id) ON DELETE
CASCADE,
    target_entity_id UUID NOT NULL REFERENCES memory_entities(id) ON DELETE
CASCADE,
    relation_type VARCHAR(50) NOT NULL CHECK (relation_type IN
('related_to', 'part_of', 'causes', 'caused_by', 'similar_to',
'opposite_to', 'precedes', 'follows')),
    relation_strength FLOAT DEFAULT 0.5,
    relation_metadata JSONB DEFAULT '{}',
    temporal_validity JSONB DEFAULT '{"start": null, "end": null}',
    confidence_score FLOAT DEFAULT 0.8,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- 创建索引
CREATE INDEX idx_memory_relations_source ON
memory_relations(source_entity_id);
CREATE INDEX idx_memory_relations_target ON
memory_relations(target_entity_id);
CREATE INDEX idx_memory_relations_type ON memory_relations(relation_type);
CREATE INDEX idx_memory_relations_strength ON
memory_relations(relation_strength DESC);
CREATE INDEX idx_memory_relations_created_at ON memory_relations(created_at
DESC);
-- 防止重复关系
CREATE UNIQUE INDEX idx_memory_relations_unique ON
memory_relations(source_entity_id, target_entity_id, relation_type);
```

知识图谱表 (knowledge_graphs)

```sql
-- 创建知识图谱表
CREATE TABLE knowledge_graphs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    project_id UUID NOT NULL REFERENCES projects(id) ON DELETE CASCADE,
    graph_name VARCHAR(100) NOT NULL,
    graph_version INTEGER DEFAULT 1,
    graph_structure JSONB DEFAULT '{}',
    entity_count INTEGER DEFAULT 0,
    relation_count INTEGER DEFAULT 0,
    evolution_config JSONB DEFAULT '{"auto_evolve": true,
"evolution_threshold": 0.7}',
    last_evolved TIMESTAMP WITH TIME ZONE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- 创建索引
CREATE INDEX idx_knowledge_graphs_project_id ON
knowledge_graphs(project_id);
CREATE UNIQUE INDEX idx_knowledge_graphs_project_unique ON
knowledge_graphs(project_id);
CREATE INDEX idx_knowledge_graphs_version ON
knowledge_graphs(graph_version);
```

图谱演化日志表 (graph_evolution_logs)

```sql
-- 创建图谱演化日志表
CREATE TABLE graph_evolution_logs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    graph_id UUID NOT NULL REFERENCES knowledge_graphs(id) ON DELETE
CASCADE,
    evolution_type VARCHAR(50) NOT NULL CHECK (evolution_type IN
('entity_added', 'entity_removed', 'relation_added', 'relation_removed',
'entity_updated', 'relation_updated', 'graph_restructured')),
    before_state JSONB,
    after_state JSONB,
    evolution_reason TEXT,
    impact_score FLOAT DEFAULT 0.0,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- 创建索引
CREATE INDEX idx_graph_evolution_logs_graph_id ON
graph_evolution_logs(graph_id);
CREATE INDEX idx_graph_evolution_logs_type ON
```

```
graph_evolution_logs(evolution_type);
CREATE INDEX idx_graph_evolution_logs_created_at ON
graph_evolution_logs(created_at DESC);
CREATE INDEX idx_graph_evolution_logs_impact ON
graph_evolution_logs(impact_score DESC);
```

初始化数据

```
-- 插入示例用户
INSERT INTO users (email, password_hash, name, plan) VALUES
('admin@supermemory.ai', '$2b$10$example_hash', 'Admin User',
'enterprise'),
('demo@example.com', '$2b$10$example_hash', 'Demo User', 'pro');

-- 插入示例项目
INSERT INTO projects (user_id, name, description, config) VALUES
((SELECT id FROM users WHERE email = 'demo@example.com'),
 'Demo Chat Bot',
 'A demonstration chatbot with memory capabilities',
 '{"model": "gpt-4", "memory_depth": 10, "temperature": 0.7}');

-- 插入默认分段配置
INSERT INTO segmentation_configs (project_id, max_chunk_size, overlap_size,
segmentation_strategy, similarity_threshold) VALUES
((SELECT id FROM projects WHERE name = 'Demo Chat Bot'),
 512, 50, 'semantic', 0.7);

-- 插入默认Token配置
INSERT INTO token_configs (project_id, max_context_tokens, reserved_tokens,
auto_optimization) VALUES
((SELECT id FROM projects WHERE name = 'Demo Chat Bot'),
 4096, 1000, true);

-- 插入示例API密钥
INSERT INTO api_keys (project_id, key_hash, name, permissions) VALUES
((SELECT id FROM projects WHERE name = 'Demo Chat Bot'),
 '$2b$10$demo_api_key_hash',
 'Demo API Key',
 '{"read": true, "write": true, "admin": false}');
```

6.2.3 RAG知识库融合数据表

RAG知识源表 (rag_knowledge_sources)

```
-- 创建RAG知识源表
CREATE TABLE rag_knowledge_sources (
```

```
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    project_id UUID NOT NULL REFERENCES projects(id) ON DELETE CASCADE,
    name VARCHAR(100) NOT NULL,
    source_type VARCHAR(50) NOT NULL CHECK (source_type IN
('elasticsearch', 'chroma', 'weaviate', 'local_files', 'database', 'api')),
    config JSONB NOT NULL DEFAULT '{}',
    status VARCHAR(20) DEFAULT 'disconnected' CHECK (status IN
('connected', 'disconnected', 'error', 'syncing')),
    document_count INTEGER DEFAULT 0,
    last_sync TIMESTAMP WITH TIME ZONE,
    sync_config JSONB DEFAULT '{"auto_sync": false, "sync_interval": "1h",
"batch_size": 100}',
    search_config JSONB DEFAULT '{"max_results": 10, "threshold": 0.7,
"boost_factor": 1.0}',
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- 创建索引
CREATE INDEX idx_rag_knowledge_sources_project_id ON
rag_knowledge_sources(project_id);
CREATE INDEX idx_rag_knowledge_sources_type ON
rag_knowledge_sources(source_type);
CREATE INDEX idx_rag_knowledge_sources_status ON
rag_knowledge_sources(status);
CREATE INDEX idx_rag_knowledge_sources_last_sync ON
rag_knowledge_sources(last_sync DESC);
```

RAG搜索日志表 (rag_search_logs)

```
-- 创建RAG搜索日志表
CREATE TABLE rag_search_logs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    source_id UUID NOT NULL REFERENCES rag_knowledge_sources(id) ON DELETE
CASCADE,
    message_id UUID REFERENCES messages(id) ON DELETE SET NULL,
    query TEXT NOT NULL,
    search_params JSONB DEFAULT '{}',
    results_count INTEGER DEFAULT 0,
    search_time_ms FLOAT DEFAULT 0.0,
    performance_metrics JSONB DEFAULT '{}',
    error_message TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- 创建索引
```

```
CREATE INDEX idx_rag_search_logs_source_id ON rag_search_logs(source_id);
CREATE INDEX idx_rag_search_logs_message_id ON rag_search_logs(message_id);
CREATE INDEX idx_rag_search_logs_created_at ON rag_search_logs(created_at
DESC);
CREATE INDEX idx_rag_search_logs_search_time ON
rag_search_logs(search_time_ms);
CREATE INDEX idx_rag_search_logs_results_count ON
rag_search_logs(results_count DESC);

-- 创建分区表（按月分区）
CREATE TABLE rag_search_logs_y2024m01 PARTITION OF rag_search_logs
FOR VALUES FROM ('2024-01-01') TO ('2024-02-01');
```

RAG同步日志表 (rag_sync_logs)

```
-- 创建RAG同步日志表
CREATE TABLE rag_sync_logs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    source_id UUID NOT NULL REFERENCES rag_knowledge_sources(id) ON DELETE
CASCADE,
    sync_type VARCHAR(50) NOT NULL CHECK (sync_type IN ('full',
'incremental', 'manual', 'scheduled')),
    sync_status VARCHAR(20) NOT NULL CHECK (sync_status IN ('running',
'completed', 'failed', 'cancelled')),
    documents_processed INTEGER DEFAULT 0,
    documents_added INTEGER DEFAULT 0,
    documents_updated INTEGER DEFAULT 0,
    documents_deleted INTEGER DEFAULT 0,
    sync_metadata JSONB DEFAULT '{}',
    error_message TEXT,
    sync_start TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    sync_end TIMESTAMP WITH TIME ZONE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- 创建索引
CREATE INDEX idx_rag_sync_logs_source_id ON rag_sync_logs(source_id);
CREATE INDEX idx_rag_sync_logs_sync_type ON rag_sync_logs(sync_type);
CREATE INDEX idx_rag_sync_logs_sync_status ON rag_sync_logs(sync_status);
CREATE INDEX idx_rag_sync_logs_sync_start ON rag_sync_logs(sync_start
DESC);
CREATE INDEX idx_rag_sync_logs_documents_processed ON
rag_sync_logs(documents_processed DESC);
```

RAG搜索结果表 (rag_search_results)

```sql
-- 创建RAG搜索结果表
CREATE TABLE rag_search_results (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    message_id UUID NOT NULL REFERENCES messages(id) ON DELETE CASCADE,
    source_id UUID NOT NULL REFERENCES rag_knowledge_sources(id) ON DELETE
CASCADE,
    result_id VARCHAR(255) NOT NULL, -- 外部系统中的文档ID
    content TEXT NOT NULL,
    score FLOAT NOT NULL DEFAULT 0.0,
    fusion_score FLOAT NOT NULL DEFAULT 0.0,
    metadata JSONB DEFAULT '{}',
    result_type VARCHAR(50) DEFAULT 'document' CHECK (result_type IN
('document', 'chunk', 'entity', 'relation')),
    result_timestamp TIMESTAMP WITH TIME ZONE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- 创建索引
CREATE INDEX idx_rag_search_results_message_id ON
rag_search_results(message_id);
CREATE INDEX idx_rag_search_results_source_id ON
rag_search_results(source_id);
CREATE INDEX idx_rag_search_results_score ON rag_search_results(score
DESC);
CREATE INDEX idx_rag_search_results_fusion_score ON
rag_search_results(fusion_score DESC);
CREATE INDEX idx_rag_search_results_result_type ON
rag_search_results(result_type);
CREATE INDEX idx_rag_search_results_created_at ON
rag_search_results(created_at DESC);
-- 复合索引用于融合搜索
CREATE INDEX idx_rag_search_results_message_fusion ON
rag_search_results(message_id, fusion_score DESC);
```

RAG知识源连接测试表 (rag_connection_tests)

```sql
-- 创建RAG知识源连接测试表
CREATE TABLE rag_connection_tests (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    source_id UUID NOT NULL REFERENCES rag_knowledge_sources(id) ON DELETE
CASCADE,
    test_type VARCHAR(50) NOT NULL CHECK (test_type IN ('manual',
'scheduled', 'health_check')),
    test_status VARCHAR(20) NOT NULL CHECK (test_status IN ('success',
'failed', 'timeout')),
```

```
    latency_ms FLOAT,
    document_count INTEGER,
    error_message TEXT,
    test_metadata JSONB DEFAULT '{}',
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- 创建索引
CREATE INDEX idx_rag_connection_tests_source_id ON
rag_connection_tests(source_id);
CREATE INDEX idx_rag_connection_tests_test_status ON
rag_connection_tests(test_status);
CREATE INDEX idx_rag_connection_tests_created_at ON
rag_connection_tests(created_at DESC);
CREATE INDEX idx_rag_connection_tests_latency ON
rag_connection_tests(latency_ms);
```

RAG融合配置表 (rag_fusion_configs)

```
-- 创建RAG融合配置表
CREATE TABLE rag_fusion_configs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    project_id UUID NOT NULL REFERENCES projects(id) ON DELETE CASCADE,
    fusion_strategy VARCHAR(50) DEFAULT 'weighted_score' CHECK
(fusion_strategy IN ('weighted_score', 'rank_fusion', 'semantic_fusion',
'temporal_fusion')),
    weight_config JSONB DEFAULT '{"supermemory": 0.6, "rag": 0.4}',
    threshold_config JSONB DEFAULT '{"min_score": 0.3, "fusion_threshold":
0.5}',
    ranking_config JSONB DEFAULT '{"max_results": 20, "diversity_factor":
0.2}',
    temporal_config JSONB DEFAULT '{"time_decay": 0.1, "recency_boost":
1.2}',
    enable_caching BOOLEAN DEFAULT true,
    cache_ttl_seconds INTEGER DEFAULT 3600,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- 创建索引
CREATE INDEX idx_rag_fusion_configs_project_id ON
rag_fusion_configs(project_id);
CREATE UNIQUE INDEX idx_rag_fusion_configs_project_unique ON
rag_fusion_configs(project_id);
```

初始化RAG相关数据

```
-- 插入示例RAG知识源
INSERT INTO rag_knowledge_sources (project_id, name, source_type, config,
status, search_config) VALUES
((SELECT id FROM projects WHERE name = 'Demo Chat Bot'),
 '企业文档库',
 'elasticsearch',
 '{"url": "http://localhost:9200", "index": "company_docs", "auth":
{"type": "basic", "username": "elastic", "password": "changeme"}}',
 'disconnected',
 '{"max_results": 15, "threshold": 0.75, "boost_factor": 1.2}'),
((SELECT id FROM projects WHERE name = 'Demo Chat Bot'),
 '技术知识库',
 'chroma',
 '{"url": "http://localhost:8000", "collection": "tech_knowledge", "auth":
{"type": "none"}}',
 'disconnected',
 '{"max_results": 10, "threshold": 0.7, "boost_factor": 1.0}');

-- 插入默认RAG融合配置
INSERT INTO rag_fusion_configs (project_id, fusion_strategy, weight_config,
threshold_config) VALUES
((SELECT id FROM projects WHERE name = 'Demo Chat Bot'),
 'weighted_score',
 '{"supermemory": 0.65, "rag": 0.35}',
 '{"min_score": 0.3, "fusion_threshold": 0.5}');
```

6.3 数据流设计

6.3.1 用户请求数据流

```
sequenceDiagram
    participant U as 用户
    participant API as API网关
    participant Auth as 认证服务
    participant Proxy as 代理服务
    participant Memory as 记忆服务
    participant Graph as 知识图谱引擎
    participant LLM as LLM服务
    participant DB as 数据库

    U->>API: 发送请求
    API->>Auth: 验证API密钥
    Auth-->>API: 返回用户信息
    API->>Proxy: 转发请求
    Proxy->>Memory: 检索相关记忆
```

```
    Memory->>Graph: 查询知识图谱
    Graph->>DB: 查询实体和关系
    DB-->>Graph: 返回图谱数据
    Graph-->>Memory: 返回关联记忆
    Memory-->>Proxy: 返回增强上下文
    Proxy->>LLM: 发送增强请求
    LLM-->>Proxy: 返回响应
    Proxy-->>API: 返回结果
    API-->>U: 返回响应
```

6.3.2 记忆存储数据流

```
sequenceDiagram
    participant Proxy as 代理服务
    participant Seg as 分段服务
    participant Vec as 向量服务
    participant Memory as 记忆服务
    participant Graph as 知识图谱引擎
    participant Analyzer as 关系分析器
    participant Builder as 语义构建器
    participant DB as 数据库

    Proxy->>Seg: 发送新对话
    Seg->>Seg: 智能分段
    Seg->>Vec: 生成向量
    Vec-->>Seg: 返回向量
    Seg->>Memory: 存储记忆片段
    Memory->>Graph: 提取实体和关系
    Graph->>Analyzer: 分析关系
    Analyzer->>Builder: 构建语义连接
    Builder->>DB: 存储实体和关系
    DB-->>Builder: 确认存储
    Builder-->>Graph: 返回构建结果
    Graph-->>Memory: 返回图谱更新
    Memory-->>Seg: 返回存储结果
    Seg-->>Proxy: 返回确认
```

6.3.3 知识图谱演化数据流

```
sequenceDiagram
    participant Memory as 记忆服务
    participant Graph as 知识图谱引擎
    participant Analyzer as 关系分析器
    participant Evolution as 演化处理器
    participant Insight as 洞察生成器
```

```
    participant DB as 数据库

    Memory->>Graph: 新记忆片段
    Graph->>Analyzer: 分析实体关系
    Analyzer->>Analyzer: 计算关系强度
    Analyzer->>Evolution: 触发演化
    Evolution->>Evolution: 评估演化条件
    Evolution->>DB: 查询历史状态
    DB-->>Evolution: 返回历史数据
    Evolution->>Insight: 生成洞察
    Insight->>Insight: 识别模式
    Insight->>DB: 更新图谱结构
    DB-->>Insight: 确认更新
    Insight-->>Evolution: 返回洞察结果
    Evolution-->>Graph: 返回演化完成
    Graph-->>Memory: 返回图谱状态
```

6.4 部署架构

6.4.1 容器化部署

```yaml
# docker-compose.yml
version: '3.8'

services:
  api-gateway:
    build: ./services/api-gateway
    ports:
      - "8080:8080"
    environment:
      - NODE_ENV=production
    depends_on:
      - redis
      - postgres
      - neo4j

  proxy-service:
    build: ./services/proxy
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=production
      - DATABASE_URL=postgresql://user:pass@postgres:5432/supermemory
      - REDIS_URL=redis://redis:6379
      - NEO4J_URL=bolt://neo4j:7687
    depends_on:
```

```
    - postgres
    - redis
    - qdrant
    - neo4j

memory-service:
  build: ./services/memory
  ports:
    - "3001:3001"
  environment:
    - NODE_ENV=production
    - DATABASE_URL=postgresql://user:pass@postgres:5432/supermemory
    - VECTOR_DB_URL=http://qdrant:6333
    - NEO4J_URL=bolt://neo4j:7687
  depends_on:
    - postgres
    - qdrant
    - neo4j

segmentation-service:
  build: ./services/segmentation
  ports:
    - "3002:3002"
  environment:
    - NODE_ENV=production
    - DATABASE_URL=postgresql://user:pass@postgres:5432/supermemory

token-service:
  build: ./services/token
  ports:
    - "3003:3003"
  environment:
    - NODE_ENV=production
    - DATABASE_URL=postgresql://user:pass@postgres:5432/supermemory

postgres:
  image: postgres:15
  environment:
    - POSTGRES_DB=supermemory
    - POSTGRES_USER=user
    - POSTGRES_PASSWORD=pass
  volumes:
    - postgres_data:/var/lib/postgresql/data
    - ./sql/init.sql:/docker-entrypoint-initdb.d/init.sql
  ports:
    - "5432:5432"
```

```
  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    volumes:
      - redis_data:/data

  qdrant:
    image: qdrant/qdrant:latest
    ports:
      - "6333:6333"
      - "6334:6334"
    volumes:
      - qdrant_data:/qdrant/storage

  neo4j:
    image: neo4j:5.15
    ports:
      - "7474:7474"
      - "7687:7687"
    environment:
      - NEO4J_AUTH=neo4j/password
      - NEO4J_PLUGINS='["apoc", "graph-data-science"]'
      - NEO4J_dbms_security_procedures_unrestricted=gds.*,apoc.*
      - NEO4J_dbms_security_procedures_allowlist=gds.*,apoc.*
    volumes:
      - neo4j_data:/data
      - neo4j_logs:/logs
      - neo4j_import:/var/lib/neo4j/import
      - neo4j_plugins:/plugins

volumes:
  postgres_data:
  redis_data:
  qdrant_data:
  neo4j_data:
  neo4j_logs:
  neo4j_import:
  neo4j_plugins:
```

6.5 性能优化

6.5.1 数据库优化

- **关系型数据库优化**

    - 为常用查询字段建立复合索引
    - 使用部分索引优化特定查询
    - 定期分析和优化查询计划
    - 连接池配置优化

- **图数据库优化**
    - 为实体和关系创建复合索引
    - 使用标签和属性索引
    - 优化图查询模式
    - 内存配置调优

- **分区策略**
    - 按时间分区大表（如消息表、使用日志表）
    - 按用户ID分区记忆数据
    - 分区表自动维护
    - 图数据库分层存储

- **缓存策略**
    - Redis缓存热点数据
    - 查询结果缓存
    - 向量检索结果缓存
    - 图结构缓存

6.5.2 向量检索优化

- **索引优化**
    - 使用HNSW算法进行近似最近邻搜索
    - 多层级索引结构
    - 动态索引更新
    - 实体嵌入向量优化

- **量化优化**
    - 向量量化减少存储空间
    - 乘积量化提高检索速度

    - 二进制量化优化内存使用
    - 关系向量压缩

- **并行化**
    - 多线程向量检索
    - 分布式向量搜索
    - GPU加速计算
    - 图算法并行化

6.5.3 知识图谱优化

- **图结构优化**
    - 实体去重和合并
    - 关系权重动态调整
    - 图分区减少查询范围
    - 层级化图组织

- **查询优化**
    - Cypher查询优化
    - 模式匹配索引
    - 图算法缓存
    - 预计算路径

- **演化优化**
    - 增量式图更新
    - 批量关系处理
    - 异步图演化
    - 内存中图构建

6.5.4 系统优化

- **负载均衡**
    - Nginx反向代理
    - 多实例服务部署
    - 动态负载分发

    - 图数据库集群

- **异步处理**
    - 消息队列处理耗时操作
    - 异步记忆存储
    - 批量处理优化
    - 图演化异步处理

- **CDN加速**
    - 静态资源CDN分发
    - API响应缓存
    - 全球节点部署
    - 图数据缓存同步

6.6 安全设计

6.6.1 认证与授权

- **JWT Token认证**
    - 短期访问令牌
    - 长期刷新令牌
    - 令牌撤销机制

- **API密钥管理**
    - 密钥哈希存储
    - 细粒度权限控制
    - 密钥轮换策略

- **多因素认证**
    - 可选MFA支持
    - 生物识别集成
    - 设备信任管理

6.6.2 数据安全

- **数据加密**
    - 传输层TLS 1.3

    - 静态数据AES-256加密
    - 敏感字段额外加密

- **隐私保护**
    - 数据脱敏处理
    - 匿名化存储
    - GDPR合规设计

- **访问控制**
    - 基于角色的访问控制(RBAC)
    - 属性基访问控制(ABAC)
    - 动态权限评估

6.6.3 系统安全

- **网络安全**
    - WAF防护
    - DDoS缓解
    - 网络隔离

- **应用安全**
    - SQL注入防护
    - XSS防护
    - CSRF防护

- **监控与审计**
    - 安全事件监控
    - 访问日志审计
    - 异常行为检测

6.7 监控与运维

6.7.1 系统监控

- **基础设施监控**
    - CPU/内存/磁盘监控
    - 网络流量监控

    - 服务健康检查

- **应用性能监控**
    - API响应时间
    - 错误率统计
    - 业务指标监控

- **数据库监控**
    - 查询性能监控
    - 连接池状态
    - 索引使用情况

6.7.2 日志管理

- **结构化日志**
    - JSON格式日志
    - 分布式链路追踪
    - 错误堆栈收集

- **日志聚合**
    - ELK Stack集成
    - 实时日志分析
    - 日志压缩归档

- **告警机制**
    - 多级别告警策略
    - 智能告警降噪
    - 告警自动恢复

6.7.3 运维自动化

- **CI/CD流程**
    - 自动化构建部署
    - 蓝绿部署策略
    - 回滚机制

- **配置管理**
  - 集中配置中心
  - 配置版本控制
  - 动态配置更新
- **容量规划**
  - 自动扩缩容
  - 资源预测分析
  - 成本优化建议

6.8 扩展性设计

6.8.1 水平扩展

- **微服务架构**
  - 服务解耦设计
  - 独立部署扩展
  - 服务发现机制
- **数据库分片**
  - 用户级分片策略
  - 地理位置分片
  - 读写分离
- **缓存集群**
  - Redis Cluster
  - 缓存分区策略
  - 热点数据分布

6.8.2 垂直扩展

- **资源优化**
  - 内存使用优化
  - CPU密集型任务优化
  - I/O性能调优
- **算法优化**

    - 时间复杂度优化
    - 空间复杂度优化
    - 并行算法设计

6.8.3 功能扩展

- **插件架构**
    - 动态插件加载
    - 插件生命周期管理
    - 插件间通信
- **API扩展**
    - GraphQL API
    - Webhook支持
    - 自定义端点
- **集成能力**
    - 第三方系统集成
    - 数据导入导出
    - 标准协议支持

6.9 高可用设计

6.9.1 故障转移

- **主备切换**
    - 自动故障检测
    - 快速主备切换
    - 数据一致性保证
- **多活架构**
    - 异地多活部署
    - 数据同步机制
    - 流量调度策略
- **降级策略**
    - 服务降级机制

    - 熔断保护
    - 限流策略

6.9.2 数据备份

- **备份策略**
    - 全量备份
    - 增量备份
    - 差异备份
- **恢复机制**
    - 点时间恢复
    - 灾难恢复
    - 数据一致性校验
- **备份验证**
    - 定期恢复测试
    - 备份完整性检查
    - 恢复时间评估

6.9.3 容灾设计

- **地理容灾**
    - 多地域部署
    - 跨地域复制
    - 地域级故障切换
- **数据容灾**
    - 数据冗余存储
    - 跨区域备份
    - 数据一致性协议
- **网络容灾**
    - 多线路接入
    - 网络故障切换

    - CDN容灾备份

6.10 成本优化

6.10.1 资源成本

- **计算资源优化**
    - 按需计算资源
    - 预留实例策略
    - 竞价实例使用
- **存储成本优化**
    - 数据分层存储
    - 冷热数据分离
    - 数据压缩策略
- **网络成本优化**
    - CDN流量优化
    - 跨区域流量调度
    - 带宽利用率提升

6.10.2 运营成本

- **自动化运维**
    - 自动化部署
    - 自动化监控
    - 自动化故障处理
- **资源调度**
    - 智能资源分配
    - 负载均衡优化
    - 能效比优化
- **成本监控**
    - 实时成本监控
    - 成本趋势分析

    - 成本优化建议

6.10.3 技术成本

- **开源技术**
    - 开源组件优先
    - 自主可控技术
    - 技术债务管理
- **标准化**
    - 技术标准统一
    - 组件标准化
    - 流程标准化
- **效率提升**
    - 开发效率优化
    - 测试效率提升
    - 部署效率改进