Weaviate 向量数据库简介

> Weaviate 是一个开源的向量搜索引擎和向量数据库，专为存储、索引和查询高维向量数据而设计。它在 RAG（检索增强生成）架构中扮演着核心角色，为AI应用提供强大的语义搜索能力。

1. 核心概念与架构

向量数据存储

- **高维向量支持**：存储和索引数百到数千维的向量数据
- **对象与向量关联**：将业务对象与其向量表示进行关联存储
- **模式定义**：支持灵活的数据模式定义和属性管理
- **多租户架构**：支持多个独立的数据集合（Classes）

向量索引技术

> **HNSW 算法**：采用分层可导航小世界图算法，提供高效的近似最近邻搜索，在准确性和性能之间取得平衡。

2. 主要功能特性

多模态数据支持

- **文本数据**：文档、段落、句子的向量化存储
- **图像数据**：图像特征向量和元数据存储
- **音频数据**：音频特征向量的索引和搜索
- **结构化数据**：传统关系型数据与向量数据的混合存储

GraphQL API

```
{ Get { Article( nearText: { concepts: ["人工智能", "机器学习"] distance: 0.7 } limit: 10 )
{ title content _additional { distance certainty } } } }
```

RESTful API

```
POST /v1/objects { "class": "Document", "properties": { "title": "Spring AI 教程",
"content": "Spring AI 是用于构建AI应用的框架...", "category": "技术文档" }, "vector": [0.123,
-0.456, 0.789, ...] }
```

3. 在 RAG 架构中的作用

知识库构建

- 文档分块存储
- 向量化索引
- 元数据管理
- 增量更新支持

语义检索

- 相似度搜索
- 混合搜索（向量+关键词）
- 多条件过滤
- 排序和分页

RAG 工作流程

1. **文档摄取**：将文档分块并转换为向量存储到 Weaviate
2. **查询处理**：用户查询转换为向量表示
3. **相似度检索**：在 Weaviate 中搜索最相关的文档片段
4. **上下文构建**：将检索结果作为上下文
5. **生成增强**：LLM 基于检索到的上下文生成回答

4. 技术优势

性能优势

- **毫秒级查询**：HNSW 索引提供快速的相似度搜索
- **水平扩展**：支持分布式部署和数据分片
- **内存优化**：高效的内存使用和缓存策略
- **并发处理**：支持高并发读写操作

易用性优势

- **模块化设计**：内置向量化模块，支持多种 ML 模型
- **自动向量化**：文本和图像的自动向量化处理
- **丰富的客户端**：Python、JavaScript、Go、Java 等语言支持
- **Cloud 服务**：提供托管服务，简化部署和维护

5. 集成示例

Python 客户端

```
import weaviate # 连接到 Weaviate 实例 client = weaviate.Client("http://localhost:8080") # 创建数据模式 schema = { "class": "Document", "vectorizer": "text2vec-transformers", "properties": [ {"name": "title", "dataType": ["string"]}, {"name": "content", "dataType": ["text"]}, {"name": "category", "dataType": ["string"]} ] } client.schema.create_class(schema) # 添加文档 doc = { "title": "Weaviate 简介", "content": "Weaviate 是一个向量数据库...", "category": "技术文档" } client.data_object.create(doc, "Document") # 语义搜索 result = client.query.get("Document", ["title", "content"]) \ .with_near_text({"concepts": ["向量数据库"]}) \ .with_limit(5) \ .do()
```

Spring AI 集成

```
# application.yml spring: ai: vectorstore: weaviate: url: http://localhost:8080 api-key: ${WEAVIATE_API_KEY} object-class: Document consistency-level: ONE # Java 代码 @Autowired private VectorStore vectorStore; public void addDocument(String content) { Document document = new Document(content); vectorStore.add(List.of(document)); } public List<Document> searchSimilar(String query) { return vectorStore.similaritySearch( SearchRequest.query(query).withTopK(5) ); }
```

6. 部署方案

本地部署

```
# Docker 部署 docker run -d \ --name weaviate \ -p 8080:8080 \ -e QUERY_DEFAULTS_LIMIT=25 \ -e AUTHENTICATION_ANONYMOUS_ACCESS_ENABLED=true \ -e PERSISTENCE_DATA_PATH='/var/lib/weaviate' \ -e DEFAULT_VECTORIZER_MODULE='none' \ semitechnologies/weaviate:latest # Docker Compose version: '3.4' services: weaviate: image: semitechnologies/weaviate:latest ports: - "8080:8080" environment: QUERY_DEFAULTS_LIMIT: 25 AUTHENTICATION_ANONYMOUS_ACCESS_ENABLED: 'true' PERSISTENCE_DATA_PATH: '/var/lib/weaviate' volumes: - weaviate_data:/var/lib/weaviate
```

云服务部署

- **Weaviate Cloud Services (WCS)**：官方托管服务
- **Kubernetes**：使用 Helm Chart 部署
- **AWS/GCP/Azure**：云平台部署支持
- **Embedded Weaviate**：嵌入式部署模式

7. 最佳实践

数据建模建议

- 合理设计 Class 结构，避免过度复杂化

- 选择合适的向量化模型和维度
- 设置合适的属性索引策略
- 考虑数据的增长和查询模式

性能优化建议

- 调整 HNSW 参数以平衡性能和准确性
- 使用批量操作提高数据导入效率
- 监控内存使用和查询延迟
- 合理设置缓存和连接池参数

8. 应用场景

- **智能客服**：基于企业知识库的问答系统
- **文档检索**：大规模文档的语义搜索
- **推荐系统**：基于内容相似度的推荐
- **图像搜索**：以图搜图功能实现
- **代码搜索**：语义化的代码片段检索
- **多语言搜索**：跨语言的语义理解和匹配

> **总结**：Weaviate 作为现代向量数据库的优秀代表，在 RAG 架构中提供了强大的语义搜索能力。其高性能、易用性和丰富的功能特性，使其成为构建智能应用的理想选择。通过与 Spring AI 等框架的深度集成，开发者可以快速构建生产级的 AI 应用。