第7章：嵌入模型（Embedding Models）与向量化存储

🎯 目标：理解文本向量化原理，为 RAG 打基础

在前几章中，我们让 AI 具备了“对话”和“行动”的能力。但从 AI 应用的完整闭环来看，还有一个关键环节：**理解与记忆**。

大模型的知识是静态的、训练时固定的。如果我们希望 AI 能“读懂”你的公司文档、产品手册、用户协议，就必须让它具备**理解新知识**的能力。

本章的核心就是 **嵌入模型（Embedding Models）** —— 它能将文本、图像等内容转化为数学向量（即“向量化”），从而使 AI 系统能够计算语义相似度、实现语义搜索，为后续的 **RAG（检索增强生成）** 系统打下坚实基础。

7.1 什么是 Embedding？它在 AI 中的作用

根据 Spring AI 官方文档，**Embeddings 是文本、图像或视频的数值化表示，用于捕捉输入之间的语义关系。**

🔍 通俗理解：AI 的“语义坐标系”

想象一下，我们有一个巨大的“语义空间”，每个词或句子都被表示为一个点（向量）。语义越接近的词，在空间中的距离就越近。

例如：

- “猫”和“狗”的向量距离很近（都是宠物）
- “飞机”和“火车”距离较近（都是交通工具）
- “猫”和“飞机”距离很远

这种将文本转化为数字向量的过程，就叫 **嵌入（Embedding）**。

数学/几何中的向量

在最简单的层面上，一个**向量**就是一个具有**大小**和**方向**的量。物理学科上叫做矢量

- **例子**：速度就是一个向量。如果你说“一辆车以 60 公里/小时的速度行驶”，这只是一个标量

（只有大小）。但如果你说“一辆车以 60 公里/小时的速度向**北**行驶”，这就是一个向量（大小是 60，方向是北）。

- **可视化**：在几何中，向量通常被画成一条带箭头的线段。线段的长度表示大小，箭头的指向表示方向。
- **坐标表示**：在二维平面中，一个向量可以用一对数字 (x, y) 来表示，比如 (3, 4)。这表示从原点 (0,0) 出发，指向点 (3,4) 的箭头。在三维空间，就是 (x, y, z)

计算机科学中的向量

在计算机科学中，**向量**的含义被泛化了。它本质上就是一个**一维数组**，即**一列有序的数字**。

- **例子**：`[1.5, -0.2, 3.14, 42.0]` 就是一个包含 4 个数字的向量。
- **维度的含义**：这个向量的**维度**就是它包含的数字个数。上面的例子是 4 维向量。在机器学习和AI中，我们经常会处理几百、几千甚至更高维度的向量。

在计算机的世界里，我们可以用这样一个数字列表（向量）来**表示任何东西**。每个数字可以被看作是描述该事物的一个**特征**或**属性**。

用向量表示一杯饮料：

- 特征1：甜度 (0 不甜，10 非常甜) –> 7
- 特征2：酸度 (0 不酸，10 非常酸) –> 2
- 特征3：温度 (0 冰，10 烫) –> 1
- 特征4：咖啡因含量 (毫克) –> 90

那么，这杯饮料就可以表示为一个 4 维向量：`[7, 2, 1, 90]`。这杯饮料很可能是一杯冰甜咖啡。

向量Embedding

向量嵌入 是“向量”概念的一个非常强大和重要的应用。它是一种**特殊类型的向量**，专门用于在计算机中**表示复杂对象**（如词语、图片、声音、用户、商品等），并捕捉其**内在含义**或**语义**。

你可以把向量嵌入理解为一个对象的**“数字指纹”**或**“DNA序列”**。

向量嵌入的强大之处在于，它不仅仅是随机的一堆数字，而是通过复杂的机器学习模型（如 Word2Vec, Transformer 等）学习得到的。这些数字的排列方式具有以下关键特性：

1. **语义相似性**
    - 含义相近的对象，它们的向量在空间中的**距离会很近**。

- **例子**：“猫”和“狗”都是宠物，它们的向量距离会很近；而“猫”和“电脑”的向量距离会很远。

2. **关系类比**
   - 向量空间中可以捕捉到类比关系。最著名的例子是：
     - **“国王”的向量 – “男人”的向量 + “女人”的向量 ≈ “女王”的向量**
   - 这意味着词语之间的语义关系（如“性别关系”）被编码在了向量的数学运算中。

向量嵌入是如何工作的？（以词嵌入为例）

想象一下，我们要把词语映射到一个 3 维空间（实际中维度更高，如300维、768维），以便可视化：

1. **初始状态**：一开始，计算机随机给每个词分配一个位置（一个向量）。
2. **学习过程**：模型阅读海量文本（如维基百科）。它学习一个原则：**“出现在相似上下文中的词语，具有相似的含义”**。
   - 例如，“苹果”和“香蕉”经常出现在“吃”、“水果”、“甜”等词语附近。
   - 而“苹果”和“微软”可能出现在“公司”、“技术”等词语附近。
3. **最终结果**：经过学习，模型会调整每个词的向量位置。
   - “苹果”（水果）和“香蕉”、“橘子”的向量会聚集在一起。
   - “苹果”（公司）和“微软”、“谷歌”的向量会聚集在另一处。
   - “水果”这个大类会和“蔬菜”等大类在更高层级上靠近。

最终，我们得到了一个“语义地图”，每个词都是这张地图上的一个点（即一个高维向量）。

✅ Embedding 的核心价值

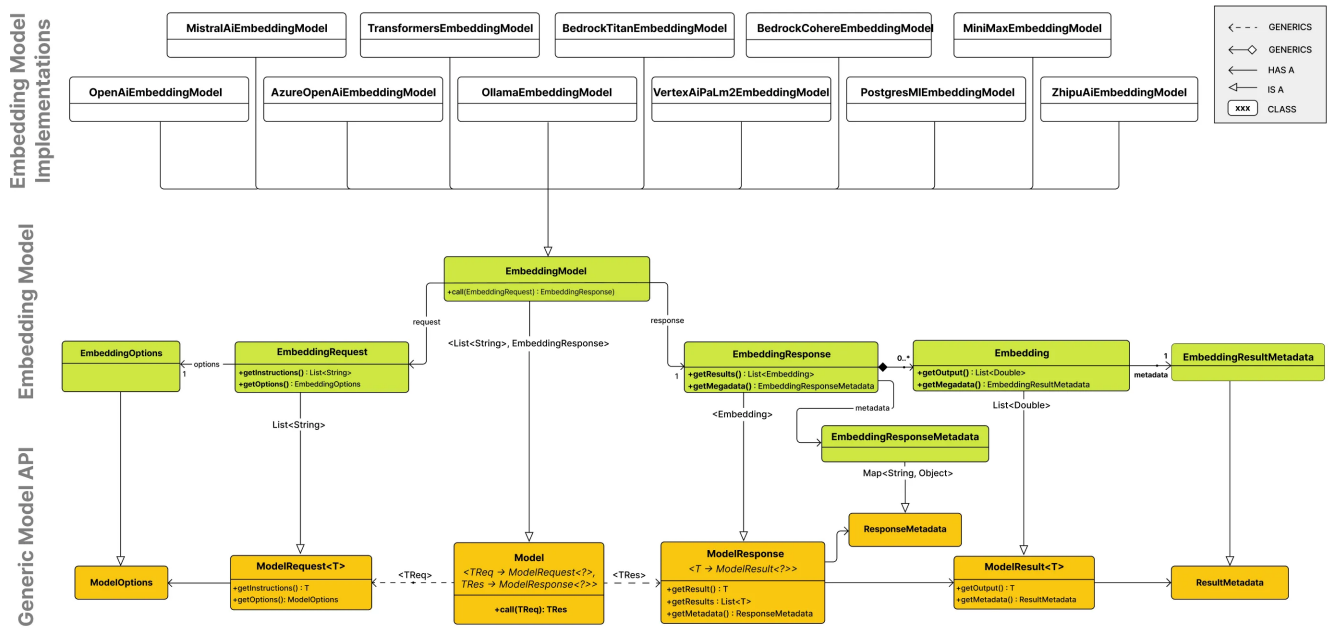
作用	说明
语义搜索	搜索“轻便的户外背包”也能匹配“登山用小容量背包”
自然语言处理	搜索引擎（理解你的查询意图）、机器翻译、智能客服、情感分析。
图像识别	将图片转换为向量，然后寻找相似的图片。
推荐系统	根据用户兴趣向量推荐相似内容
RAG 基础	检索最相关的知识片段，供大模型参考生成答案

7.2 Spring AI 中的 `Embedding` 接口与类

下面这个表格整理了Spring AI中与Embedding相关的核心接口与类：

接口/类	主要职责
`EmbeddingModel`	嵌入模型的核心接口，定义了文本向量化的能力
`EmbeddingRequest`	封装了向嵌入模型发出的请求，包含待处理的文本和模型配置选项
`EmbeddingResponse`	封装了嵌入模型的返回结果
`Embedding`	代表一段文本对应的向量化结果，即浮点数列表
`EmbeddingOptions`	配置嵌入模型参数（如模型名称、温度值等）的接口，不同模型提供商有其具体实现

来源于官方网站的类结构图：



Spring AI 提供了统一的 `EmbeddingModel` 接口，屏蔽了不同模型厂商的差异，实现**可移植性**和**易用性**。

📦 核心接口定义

```java
public interface EmbeddingModel extends Model<EmbeddingRequest, EmbeddingResponse> {

    //
    @Override
    EmbeddingResponse call(EmbeddingRequest request);

    // 核心方法：将文本转为向量
    float[] embed(String text);

    //将从文档中抽取出来的Document转换为向量，其中Document对象是在Spring AI文本抽取定义
    float[] embed(Document document);

    // 批量处理
    List<float[]> embed(List<String> texts);

    // 返回完整响应（含元数据）
    EmbeddingResponse embedForResponse(List<String> texts);

    // 获取向量维度
    int dimensions();
}
```

下面这个表格整理了 Spring AI 支持的主要嵌入模型类别及其代表：

模型类别	代表模型/提供商
云服务厂商模型	OpenAI (`text-embedding-3-small/large`), Azure OpenAI, Google Vertex AI (Gemini Embeddings), 阿里云 (百炼平台 `text-embedding-v3`), 腾讯云 (混元 Embeddings)
开源/本地部署模型	Ollama (本地运行各种LLM模型并生成嵌入), Transformers (ONNX 格式的预训练模型)

💡 国内开发者推荐使用 通义千问 或 Ollama 本地部署，避免网络问题。

7.3 使用Spring AI Embedding

使用 Spring AI 的嵌入功能通常包含以下几个步骤，这里以配置阿里云百炼平台的模型为例，其他模型的接入方式类似。

1. 添加项目依赖

本章节将会使用 阿里云百炼平台的Embedding模型，所以要引入百炼平台的SDK，前面几个章节我们一直在使用它

XML

```xml
<!-- chapter05/pom.xml -->
<dependency>
    <groupId>com.alibaba.cloud.ai</groupId>
    <artifactId>spring-ai-alibaba-starter-dashscope</artifactId>
</dependency>
<!-- 本章会编写测试用例进行测试 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

2. 配置模型参数

在项目 `application.yml` 中配置embedding模型. 主要指定了模型名称:

`spring.ai.dashscope.embedding.options.model`

```yaml
# chapter05/src/main/resources/application.yml
spring:
  ai:
    dashscope:
      api-key: ${AI_BAI_LIAN_API_KEY} # 必填,在操作系统环境变量中设置这个变量后,重启IDEA才能生效。因为IDEA启动的时候会缓存这个变量
      chat:
        options:
          model: qwen-plus
          # 这个值0~1，值越大代表生成的结果随机性越强。如果是一个聊天，这个值可以大一点。如果是一些严谨的规划，则这个值可以设置小一些
          temperature: 0.7
      embedding:
        options:
          model: text-embedding-v3  # 指定使用的嵌入模型
```

3. 注入并使用EmbeddingModel

自动配置将会自动创建`EmbeddingModel`的实例对象，创建一个测试用例进行测试，看`EmbeddingModel`是否生效

```java
// chapter05/src/test/java/com/kaifamiao/chapter07/EmbeddingModelTest.java
@SpringBootTest
@Slf4j
public class EmbeddingModelTest {
    @Autowired
    private EmbeddingModel embeddingModel;

    @Test
    public void testEmbeddingModel() {

        log.info("embeddingModel:{}", embeddingModel.getClass());//class com.alibaba.cloud.ai.dashscope.embedding.DashScopeEmbeddingModel
        String  text      = "Hello, World!";
        float[] embedding = embeddingModel.embed(text);

        log.info("Embedding length:{} ", embedding.length);//1024

        for (int i = 0; i < 10; i++) {
            log.info("{} : {}", i, embedding[i]);
            /*
            0 : -0.040509745
            1 : 0.048558544
            2 : -0.06773139
            3 : -0.020919278
            4 : -0.06309953
            5 : -0.056987002
            6 : -0.033808745
            7 : -0.015091495
            8 : -0.020539619
            9 : 9.4796414E-4
             */
        }
    }
}

```

1. 通过观察日志， `EmbeddingModel` 被注入的具体的类型是 `DashScopeEmbeddingModel` ,这个Bean在自动配置中被创建。具体代码可参考 `com.alibaba.cloud.ai.autoconfigure.dashscope.DashScopeEmbeddingAutoConfiguration` 这个类的源码。
2. `Hello, World` 这段文字，被 `embed` 后，是一个长度为1024的 float类型的数组，代码中输出了前10 个数据

```java
// chapter05/src/test/java/com/kaifamiao/chapter07/EmbeddingModelTest.java
@SpringBootTest
@Slf4j
public class EmbeddingModelTest {
    @Autowired
    private EmbeddingModel embeddingModel;

    ...

    @Test
    public void callTest() {
        var input = "你好";
        EmbeddingOptions embeddingOptions = EmbeddingOptionsBuilder.builder()
                // 设定embedding模型名称
                .withModel("text-embedding-v4")//该模型默认维度为1024
                .withDimensions(128) //设定embedding模型维度
                .build();
        EmbeddingRequest  embeddingRequest = new EmbeddingRequest(List.of(input), embeddingOptions);
        EmbeddingResponse response         = embeddingModel.call(embeddingRequest);
        log.info("EmbeddingResponseMetadata:{}", response.getMetadata().getUsage());
        response.getResults().forEach(result -> {
            log.info("EmbeddingResult:{}-> {}", result.getIndex(), result.getOutput());
        });
    }
}
```

7.4 向量相似度计算原理（余弦相似度）

向量相似度是判断两个嵌入向量语义相关性的核心指标，**余弦相似度**是最常用的计算方式。

余弦相似度衡量两个向量在空间中的方向一致性，计算公式：

$$\text{similarity} = \cos(\theta) = \frac{\sum_{i=1}^{n}(A_i \times B_i)}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

- 分子：向量 A 和 B 的点积（反映方向一致性）
- 分母：向量模长的乘积（归一化操作，消除长度影响）

- 结果范围：[–1, 1]，越接近 1 表示语义越相似.
- 几何意义
  - 若两个向量方向完全相同，夹角=0，相似度 = 1
  - 若向量垂直(无关联), 即夹角=90， 相似度=0
  - 若完全相反, 即夹角 =180, 相似度 =–1

下面编写代码来计算三段文字的相似度：

1. "Spring AI 是一个用于构建 AI 应用的框架"
2. "Spring AI 帮助开发者快速集成人工智能功能"
3. "Java 是一种跨平台的编程语言"

5.4.1 编写一个Service类封装向量计算

Java

```java
// chapter07/src/main/java/com/kaifamiao/chapter07/service/MyEmbeddingService.java
@Service
public class MyEmbeddingService {
    private final EmbeddingModel embeddingModel;

    public MyEmbeddingService(EmbeddingModel embeddingModel) {
        this.embeddingModel = embeddingModel;
    }

    /**
     * 批量计算文本的嵌入向量
     *
     * @param texts 文本
     * @return 嵌入向量
     */
    public List<float[]> embed(List<String> texts) {
        return embeddingModel.embed(texts);
    }

    public double cosineSimilarity(float[] vec1, float[] vec2) {
        if (vec1.length != vec2.length) {
            throw new IllegalArgumentException("向量维度必须一致");
        }
        // 点积
        double dotProduct = 0.0;
        double norm1      = 0.0;
        double norm2      = 0.0;

        for (int i = 0; i < vec1.length; i++) {
            //点积计算：通过循环累加每个维度上的乘积累积得到点积值。
            dotProduct += vec1[i] * vec2[i];
            //模长平方计算：分别对两个向量各维度进行平方并求和，得到了各自的模长平方。
            norm1 += Math.pow(vec1[i], 2);
            norm2 += Math.pow(vec2[i], 2);
        }

        if (norm1 == 0 || norm2 == 0) {
            return 0.0; // 避免除零
        }
        // 余弦相似度计算：将点积除以两个向量的模长的乘积，得到余弦相似度值。
        return dotProduct / (Math.sqrt(norm1) * Math.sqrt(norm2));
    }
}
```

5.4.2 编写测试用例

Java

```java
// chapter07/src/test/java/com/kaifamiao/chapter07/EmbeddingModelTest.java

@SpringBootTest
@Slf4j
public class EmbeddingModelTest {
    ...
    @Autowired
    private MyEmbeddingService myEmbeddingService;
    ...
    @Test
    public void cosineSimilarityTest() {
        // 1. 文本嵌入示例
        String        text1      = "Spring AI 是一个用于构建 AI 应用的框架";
        String        text2      = "Spring AI 帮助开发者快速集成人工智能功能";
        String        text3      = "Java 是一种跨平台的编程语言";
        List<float[]> embeddings = myEmbeddingService.embed(List.of(text1, text2, text3));
        log.info("向量维度:{} ", embeddings.getFirst().length);

        // 2. 计算余弦相似度
        double similarity1_2 = myEmbeddingService.cosineSimilarity(embeddings.get(0), embeddings.get(1));
        double similarity1_3 = myEmbeddingService.cosineSimilarity(embeddings.get(0), embeddings.get(2));
        System.out.printf("text1 与 text2 相似度: %.4f%n", similarity1_2); // 应接近 1
        System.out.printf("text1 与 text3 相似度: %.4f%n", similarity1_3); // 应较低
    }
}
```

LaTeX

```latex
控制台输出:  可见相似度越高，值越大
向量维度:1024
text1 与 text2 相似度: 0.8742
text1 与 text3 相似度: 0.5774
```

7.5 向量存储(VectorStore)

在大模型驱动的 AI 应用开发中，“非结构化数据处理”与“相似性检索”是核心需求 —— 无论是问答系统需要匹配知识库文档，还是推荐系统需要找到相似商品描述，都离不开对文本、图片等非结构化数据的向量化处理。

Spring AI 作为简化 AI 应用开发的框架，通过“向量存储（Vector Store）”组件统一了不同向量数据库的操作接口，让开发者无需关注底层存储细节，即可快速实现向量的存储、检索与管理。

7.5.1 什么是向量存储？为什么需要它？

在 AI 应用中，非结构化数据（如一篇文章、一张图片）无法直接被大模型用于“相似性对比”，必须先通过“嵌入模型（Embedding Model）”将其转换为**高维向量（Embedding Vector）** —— 这些向量的数学距离（如欧氏距离、余弦相似度）能直接反映原始数据的语义相似性（比如“猫抓老鼠”和“猫咪捕捉耗子”的向量距离会非常近）。

而**向量存储**就是专门用于存储这些高维向量，并提供“相似性检索”能力的组件。它的核心价值在于：

1. **高效检索**：相比传统数据库的“精确匹配”，向量存储能快速从百万 / 千万级向量中找到与目标向量最相似的结果（如“找到与用户问题最相关的 3 篇知识库文档”）；
2. **解耦设计**：将向量存储与嵌入模型、大模型解耦，开发者可独立选择嵌入模型（如 OpenAI Embedding、Hugging Face 本地模型）和向量数据库（如 Chroma、Pinecone）；
3. **支撑 RAG 核心流程**：在“检索增强生成（RAG）”架构中，向量存储是“检索环节”的核心 —— 通过检索相似上下文补充给大模型，避免大模型“失忆”或输出错误信息。

7.5.2 向量存储核心接口

Spring AI 对向量存储的设计遵循“**抽象统一、实现灵活**”的原则，核心是通过顶层接口定义通用能力，再针对不同向量数据库提供具体实现，开发者基于接口编程即可无缝切换底层存储。

Spring AI 在 `org.springframework.ai.vectorstore` 包中定义了 `VectorStore` 接口，封装了向量存储的所有核心操作，最常用的方法包括：

方法名	作用
`add(List<Document> documents)`	将文档（含内容、元数据、向量）存入向量存储（若未传入向量，会自动调用嵌入模型生成）
`similaritySearch(String query, int topK)`	根据查询文本，检索 Top K 个最相似的文档（内部自动将 query 转为向量）

`similaritySearch(String query, int topK, Map<String, Object> filter)`	带元数据过滤的相似性检索（如“只检索标签为‘技术文档’的相似文档”）
`delete(String id)`	根据文档 ID 删除向量
`delete(List<String> ids)`	批量删除向量

其中 `Document` 是 Spring AI 定义的“数据载体”，包含以下核心字段：

```java
public class Document {
    ...
    private final String id;
    private final String text;
    private final Media media;
    private final Map<String, Object> metadata;
    @Nullable
    private final Double score;
    @JsonIgnore
    private ContentFormatter contentFormatter;
    ...
}
```

- `id`：作为文档在向量存储中的唯一标识，用于精准定位、更新或删除文档；若开发者在创建 `Document` 时手动指定（如业务系统中的文档 ID），则直接使用，若未指定，向量存储会自动生成（通常为 UUID），确保唯一性。
- `text`：文档核心文本内容；存储文档的文本信息，是向量生成的主要依据（嵌入模型会将 `text` 转换为向量）。
- `media`：多媒体内容载体。类型为Media，是SpringAI 定义的多媒体封装类，其作用是支持存储非文本类型的内容，如图片、音频、视频等，实现“多模态文档”的处理。
- `metadata`：文档元数据，存储文档的辅助信息，用于检索时的过滤、分类或溯源，常见内容包括：
  - 来源信息：`source="https://example.com/docs"`、`author="张三"`；
  - 分类标签：`category="技术文档"`、`priority="high"`；
  - 时间信息：`createTime="2024-09-30"`；
  - 自定义业务字段：`productId="P12345"`（关联业务系统 ID）

结合向量存储的过滤能力（如 `similaritySearch` 方法的 `filter` 参数），可实现“按元数据筛选 + 相似性检索”的复合查询（如“只检索 2024 年发布的‘技术文档’中与 query 相似的

内容”）。

- `score`：相似度得分，仅在“检索结果”中有效，表示当前文档与查询向量的相似度得分（值越大，相似度越高）。由向量存储在执行 `similaritySearch` 时自动计算并赋值，创建文档（`add` 操作）时无需指定。
- `contentFormatter` 内容格式化器。定义文档内容的格式化规则，用于将 `text` 和 `media` 转换为特定格式（如大模型可理解的 Prompt 片段、前端展示的 HTML 等）

若未指定，使用 `DefaultContentFormatter`，默认将 `text` 直接作为格式化结果，忽略 `media`（需自定义实现以支持多媒体格式化）。

7.5.3 SimpleVectorStore

`SimpleVectorStore` 是一个轻量级的内存向量存储实现，专为快速开发、测试和小型规模数据场景设计。它无需依赖外部数据库，所有向量和文档都存储在内存中，适合作为向量存储功能的“入门示例”或简单应用的临时存储方案。

- 配置 `SimpleVectorStore` Bean

```java
// chapter07/src/main/java/com/kaifamiao/chapter07/configuration/VectorStoreConfig.java

@Configuration
public class VectorStoreConfig {
    @Bean
    public VectorStore vectorStore(EmbeddingModel embeddingModel) {
        return SimpleVectorStore.builder(embeddingModel).build();
    }

}
```

- 实战 `VectorStore` 用法

首先在 SimpleVectorStore中保存一个商品，然后进行相似度搜索

Java

```java
// chapter07/src/test/java/com/kaifamiao/chapter07/VectorStoreTest.java
@SpringBootTest
@Slf4j
public class VectorStoreTest {

    @Autowired
    private VectorStore vectorStore;

    @Test
    public void testVectorStore() {
        log.info("vectorStore:{}", vectorStore.getClass());//org.springframework.ai.vectorstore.SimpleVectorStore

        // 1. 创建文档元数据
        Map<String, Object> metadata = Map.of(
                "category", "电子产品",
                "price", 5999.00,
                "releaseDate", "2024-01-15",
                "image", "https://gw.alicdn.com/bao/uploaded/i2/O1CN01lNEzKr1QXBHH4WsQJ_!!4611686018427381953-0-rate.jpg_960x960.jpg_.webp"
        );
        // 2. 创建文档（指定ID、文本、元数据）
        Document phoneDoc = Document.builder()
                .id("doc-123")// 自定义ID
                .text("iPhone 12 配备A14芯片，6.1英寸屏幕，支持5G网络...") // 文本内容
                //.media(productImage) // 关联图片
                .metadata(metadata) // 元数据
                .build();
        // 3. 存入向量存储
        vectorStore.add(List.of(phoneDoc));

        // 4. 检索时获取带score的结果
        List<Document> results = vectorStore.similaritySearch("推荐一款支持5G的手机");
        Document       topDoc  = results.getFirst();
        log.info("匹配的文档ID：{}", topDoc.getId());//doc-123
        log.info("匹配的文档元数据：{}", topDoc.getMetadata());//{image=https://gw.alicdn.com/bao/uploaded/i2/O1CN01lNEzKr1QXBHH4WsQJ_!!4611686018427381953-0-rate.jpg_960x960.jpg_.webp, category=电子产品, distance=0.39520738563777547, releaseDate=2024-01-15, price=5999.0}
        log.info("匹配的文档文本：{}", topDoc.getText());//iPhone 12 配备A14芯片，6.1英寸屏幕，支持5G网络...
    }
}
```

7.6 Milvus 向量数据库

`SimpleVectorStore` 是内存向量数据库。Milvus 是一款开源的分布式向量数据库，专为处理海量高维向量数据设计，广泛应用于 AI 领域的相似性检索场景。 https://milvus.io/

7.6.1 关键概念

- Collection：相当于关系数据库中的表，是存储向量和标量数据的逻辑单元
- Partition：Collection 的分区，用于数据分片和隔离
- Vector：高维向量数据，Milvus 支持 1 到 32768 维的向量
- Index：为向量建立的索引结构，用于加速相似性检索
- Entity：Milvus 中的基本数据单元，由向量字段和标量字段组成
- Metric Type：向量相似度计算方式，如欧氏距离（L2）、余弦相似度（IP）等

7.6.2 安装Milvus向量数据库

以下使用docker进行安装. 假设服务器IP地址为 `192.168.31.254`

Shell

```
wget https://github.com/milvus-io/milvus/releases/download/v2.6.0/milvus-standalone-docker-compose.yml -O docker-compose.yml

docker compose -f milvus-standalone-docker-compose.yml  up -d
```

启动 Milvus 后,名为milvus- standalone、milvus-minio 和milvus-etcd的容器启动。

- **milvus-etcd**容器不向主机暴露任何端口，并将其数据映射到当前文件夹中的**volumes/etcd**。
- **milvus-minio**容器使用默认身份验证凭据在本地为端口**9090**和**9091**提供服务，并将其数据映射到当前文件夹中的**volumes/minio**。
- **Milvus-standalone**容器使用默认设置为本地**19530**端口提供服务，并将其数据映射到当前文件夹中的**volumes/milvus**。

你还可以访问 Milvus WebUI，网址是 **`http://192.168.31.254:9091/webui/`**

7.6.3 对接Milvus向量数据库

1. 添加依赖

```xml
# chapter07/pom.xml
<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-milvus-store</artifactId>
</dependency>

 <dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-starter-vector-store-milvus</artifactId>
</dependency>

<dependency>
    <groupId>io.milvus</groupId>
    <artifactId>milvus-sdk-java</artifactId>
    <version>2.6.4</version>
</dependency>
```

2. 指定Milvus 数据库配置

`spring-ai-milvus-store` 提供了 `org.springframework.ai.vectorstore.milvus.MilvusVectorStore` 类，它实现了 `org.springframework.ai.vectorstore.VectorStore` 接口，所以只需要在在配置类中将 `SimpleVectorStore` 这个Bean替换成 `MilvusVectorStor` 即可：

`MilvusServiceClient` 用于创建于Milvus 向量数据库连接客户端。

```java
//chapter07/src/main/java/com/kaifamiao/chapter07/configuration/VectorStoreConfig.java
@Configuration
public class VectorStoreConfig {
    //    @Bean
    //    public VectorStore vectorStore(EmbeddingModel embeddingModel) {
    //        return SimpleVectorStore.builder(embeddingModel).build();
    //    }
    @Bean
    public VectorStore vectorStore(MilvusServiceClient milvusClient, EmbeddingModel embeddingModel) {
        return MilvusVectorStore.builder(milvusClient, embeddingModel)
        .collectionName("default")
        .databaseName("default")
        .indexType(IndexType.IVF_FLAT)
        .metricType(MetricType.COSINE)
        .initializeSchema(true)
        .build();
    }

    @Bean
    public MilvusServiceClient milvusClient() {
        return new MilvusServiceClient(ConnectParam.newBuilder()
                                       .withHost("192.168.31.254")
                                       .withPort(19530)
                                       .build());
    }
}
```

3. 执行测试用例

先前的测试用例：

`com.kaifamiao.chapter07.EmbeddingModelTest.testVectorStore` 不用做任何更改，因为此时注入的 `VectorStore` 就是 `MilvusVectorStor` 实例对象。