LangChain & LangChain4j

Guide détaillé en 5 pages

Version 2025

Introduction

Les modèles de langage (LLM) comme Gemini, GPT ou Claude sont puissants, mais ils deviennent réellement exploitables lorsqu'ils sont connectés à des sources de données, à une mémoire, ou à des outils. C'est précisément le rôle de **LangChain** et **LangChain4j**.

Ces frameworks permettent :

- la création d'agents intelligents,
- l'intégration de modèles de langage,
- la mise en place d'un système RAG (Retrieval-Augmented Generation),
- la connexion à des outils, API, bases vectorielles, etc.

LangChain4j reprend les concepts de LangChain (Python) et les rend disponibles pour l'écosystème Java.

LangChain : principes fondamentaux

1. Modèles de langage

LangChain unifie l'utilisation de plusieurs modèles :

- OpenAI / GPT
- Google Gemini
- Anthropic Claude
- LLaMA, Mixtral, DeepSeek (via API ou local)

Exemple (Python) :

```
from langchain_community.chat_models import ChatOpenAI

model = ChatOpenAI(model="gpt-4")
response = model.invoke("Bonjour␣!")
```

2. Embeddings

Les embeddings transforment un texte en vecteurs numériques.

Utilités :

- recherche sémantique,
- RAG,
- clustering,
- détection de similarité.

Exemple :

```
from langchain.embeddings import OpenAIEmbeddings

emb = OpenAIEmbeddings()
vector = emb.embed_query("Bonjour␣tout␣le␣monde")
```

3. RAG : Retrieval-Augmented Generation

Principe en quatre étapes :

1) Documents → **chunks** 2) Chunks → **embeddings** 3) Insertion dans une **base vectorielle** 4) Lors d'une question : → recherche des chunks pertinents → l'LLM répond en utilisant ces informations

4. Agents

Les agents prennent des décisions et appellent des outils.

```
from langchain.agents import initialize_agent, Tool

tools = [...]
agent = initialize_agent(tools, model, agent="zero-shot-react-
   description")
agent.run("Cherche␣la␣m  t  o␣et␣r sume-la.")
```

LangChain4j : la version Java

1. ChatModel

```
ChatModel model = GoogleAiGeminiChatModel.builder()
    .apiKey(System.getenv("GEMINI_API_KEY"))
    .modelName("gemini-2.5-flash")
    .build();
```

2. EmbeddingModel

```
EmbeddingModel embedding = GoogleAiEmbeddingModel.builder()
    .apiKey(System.getenv("GEMINI_API_KEY"))
    .modelName("text-embedding-004")
    .build();
```

3. Vector Store

```
EmbeddingStore<TextSegment> store = new InMemoryEmbeddingStore<>();
```

4. Ingestion

```
EmbeddingStoreIngestor ingestor = EmbeddingStoreIngestor.builder()
    .embeddingModel(embedding)
    .embeddingStore(store)
    .build();

ingestor.ingest(document);
```

5. Assistant RAG

```
Assistant assistant = AiServices.builder(Assistant.class)
    .chatModel(model)
    .contentRetriever(EmbeddingStoreContentRetriever.from(store))
    .build();
```

Comparaison LangChain vs LangChain4j

LangChain (Python) Écosystème le plus complet Support natif de nombreux modèles Beaucoup de modules avancés (Tools, Agents, RAG) Dépend du runtime Python **LangChain4j (Java)** Optimal pour les entreprises Java Intégration Spring Boot Performances élevées Encore jeune : moins de modules

Cas d'utilisation

- Chatbots intelligents connectés à des PDF
- Analyse documentaire automatisée
- Assistants pour applications Java
- Génération de code / support technique
- RAG + Vector DB dans le backend

Conclusion

LangChain et LangChain4j sont les frameworks de référence pour développer des applications IA modernes. Leur force réside dans la capacité à :

- structurer les interactions LLM,
- connecter des sources de données,
- ingérer et rechercher des documents,
- créer des agents intelligents.

Ces outils transforment un modèle de langage en un **système d'IA complet**, capable d'agir, de lire, de comprendre et de répondre de manière contextualisée.