LangChain4j

EMSI - Université Côte d'Azur
Richard Grin
Version 1.1 - 10/12/24

1

Plan du support

- LangChain
- Présentation et bases de LangChain4j
- Chaines et services
- Outils
- Modération
- Streaming
- APIs et outils autour de l'IA
- Références

2

LangChain

3

Présentation

- https://www.langchain.com/
- Framework open source pour développer des applications qui utilisent des LLM
- Couche abstraite pour travailler avec l'API de nombreux LLMs (OpenAI, Gemini, Llama, Anthropic, Mistral, ...)
- Permet de combiner des LLM avec des applications et des sources de données externes
- Permet de créer des « chaînes », des séries d'actions enchaînées les unes aux autres
- Librairie officielle pour Python et JavaScript, pas pour Java

4

Applications d'IA

- Les tâches complexes nécessitent
  - un découpage en plusieurs sous-tâches
  - l'utilisation de plusieurs types de modèles
  - leur configuration
  - l'utilisation de plusieurs types de supports (textes, images, sons,...)
  - l'apport de données spécifiques à l'utilisateur (RAG)
- LangChain facilite l'exécution de ces tâches et leur indépendance par rapport aux produits utilisés

5

Chaines

- Le cœur de LangChain
- Une chaine peut être composée de primitives ou d'autres chaines, exécutées dans un certain ordre
- C'est un pipeline qui traite une entrée et retourne un résultat

6

Principaux types de chaines

- Utilitaire : extrait une réponse d'un LLM ; par exemple LLMMathChain ajoute la possibilité à un LLM de faire des maths
- Générique : utilisé comme bloc de construction ; enchaine plusieurs étapes pour accomplir une tâche complexe ; configurable avec des paramètres

7

Exemples d'utilisation

- Permet de connecter un LLM à des sources de données privées (par exemple des fichiers PDF) pour lancer des actions, par exemple envoyer un email
- Assistant pour la météo ; quand le LLM reçoit une question qui concerne la météo, il peut interroger une API météo distante

8

Présentation et bases de LangChain4j

9

Présentation

- https://langchain4j.github.io/langchain4j/
- Adaptation de LangChain pour Java
- Standard de facto pour IA avec Java
- Fournit
  - APIs unifiées pour les LLMs et les bases de données vectorielles (pour manipuler les embeddings)
  - Boite à outils pour les prompts, la gestion de la mémoire, le RAG
  - De nombreux exemples de code pour des cas d'utilisation

10

Contenu API

- Package dev.langchain4j, avec des sous-packages chain, classification, code, data, exception, memory, model, rag, retriever, service, store, …
- 2 niveaux d'abstraction
  - Bas niveau pour manipuler les modèles, les messages, les magasins/entrepôt, les embeddings, …
  - Plus haut niveau en utilisant `AiServices` (configuration déclarative)

11

Intégration avec les LLMs

- Azure OpenAi
- DashScope
- Google AI Gemini (avec texte, image, audio, vidéo, PDF)
- Google Vertex AI Gemini
- HuggingFace
- Jlama
- LocalAI
- Mistral AI
- Ollama
- OpenAI (avec texte, image)
- …

12

Dépendances Maven - Gemini

```
<dependency>
    <groupId>dev.langchain4j</groupId>
    <artifactId>langchain4j</artifactId>
    <version>0.37.0</version>
</dependency>
<dependency>
    <groupId>dev.langchain4j</groupId>
    <artifactId>langchain4j-core</artifactId>
    <version>0.37.0</version>
</dependency>
<dependency>
    <groupId>dev.langchain4j</groupId>
    <artifactId>langchain4j-google-ai-gemini</artifactId>
    <version>0.37.0</version>
    <type>jar</type>
</dependency>
```


13

Dépendances Maven - OpenAI

```
<dependency>
    <groupId>dev.langchain4j</groupId>
    <artifactId>langchain4j</artifactId>
    <version>0.35.0</version>
</dependency>
<dependency>
    <groupId>dev.langchain4j</groupId>
    <artifactId>langchain4j-open-ai</artifactId>
    <version>0.35.0</version>
</dependency>
```

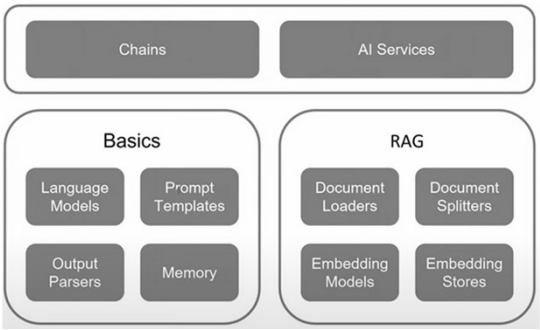

14

Clé secrète pour API

- La plupart des LLMs nécessitent une clé pour identifier l'utilisateur
- Il n'est pas bon d'écrire la valeur de la clé dans le code
- Le plus simple est d'ajouter une variable d'environnement dans l'OS et de récupérer sa valeur dans le code :
  `String cle = System.getenv("CLE_API");`

15

Composants de LangChain4j



16

Types de modèles

- Interface `ChatLanguageModel` : représente un LLM qui traite des requêtes avec un ou plusieurs `ChatMessage` en entrée et un `AiMessage` en sortie
- Interface `EmbeddingModel` : pour convertir des textes en embeddings
- Interface `ImageModel` : générateur pour images ou textes
- Interface `ModerationModel` : surveillance des contenus échangés avec les LLMs

17

Interface ChatLanguageModel (1/2)

- Package dev.langchain4j.model.chat
- Plusieurs méthodes `generate` surchargées qui génèrent la réponse du modèle
- Méthode de base

```
Response<AiMessage> generate(List<ChatMessage> messages)
```

Les messages précédents de la conversation sont aussi passés en paramètre
L'implémentation est fournie par des dépendances associées aux différents LLMs

18

Interface ChatMessage

- Package dev.langchain4j.data.message
- Représente un message échangé pendant une conversation avec un LLM
- Implémenté par 4 classes dans le même package

19

Types de ChatMessage

- `UserMessage` : message de l'application ou de l'utilisateur qui contient du texte (`String`) ou du texte et des images (`Image`)
- `AiMessage` : message généré par le LLM en réponse à un `UserMessage` ; peut contenir un texte (`String`) ou une requête pour exécuter un outil (`ToolExecutionRequest`)
- `SystemMessage` : message pour le système ; décrit le rôle du LLM, comment il doit se comporter, le style de réponse,... Souvent au début de la conversation
- `ToolExecutionResultMessage` : résultat d'une `ToolExecutionRequest`

20

Classe UserMessage

- Package dev.langchain4j.data.message
- Plusieurs façons de créer un `UserMessage` avec une `String`, un `Content` ou une liste de `Content` de différents types, et optionnellement un nom d'utilisateur :
  - constructeur
  - méthode `static from`
  - méthode `static userMessage`
- Méthodes `singleText()`, `contents()` pour retrouver le contenu du message

21

Interface Content

- Package dev.langchain4j.data.message
- Une seule méthode `ContentType type()`
- `ContentType` est une énumération dont les valeurs sont `TEXT`, `TEXT_FILE`, `PDF`, `AUDIO`, `IMAGE`, `VIDEO`
- Implémentée par `TextContent`, `TextFileContent`, `PdfFileContent`, `ImageContent`, `AudioContent`, `VideoContent`

22

Classe SystemMessage

- Package dev.langchain4j.data.message
- Souvent défini par le développeur, pas l'utilisateur
- Instructions sur le comportement, le style, le rôle du LLM

23

System et User messages

- Les LLMs et les frameworks qui les utilisent (comme LangChain4j) donnent le plus souvent la priorité aux messages système s'il y a contradiction avec des messages utilisateur

24

AIMessage

- Package dev.langchain4j.data.message
- Réponse du LLM
- Peut contenir du texte ou une requête pour exécuter un ou plusieurs outils

25

Response<T>

- Package dev.langchain4j.model.output
- Représente une réponse d'un modèle, y compris pour un modèle d'embeddings
- `T` : type de contenu généré par le modèle
  - `AiMessage` pour un `ChatLanguageModel`
  - `Embedding` pour un `EmbeddingModel`

26

Méthodes de Response<T>

- `T content()` : récupère le contenu
- `TokenUsage tokenUsage()` : récupère les statistiques sur l'usage du modèle (nombre de tokens en entrée et en sortie)
- `Map<String, Object> metadata()` : récupère les métadonnées
- `FinishReason finishReason()` : récupère la raison de l'arrêt de la génération par le LLM

27

TokenUsage

- Package dev.langchain4j.model.output
- `Integer inputTokenCount()`
- `Integer outputTokenCount()`
- `Integer totalTokenCount()`
- Possible de cumuler 2 `TokenUsage`

28

Exemple utilisation TokenUsage

```
UserMessage message = UserMessage.from("...");
Response<AiMessage> reponse = model.generate(message);

// Obtenir le nombre de tokens utilisés
TokenUsage usage = reponse.tokenUsage();
int totalTokens = usage.totalTokenCount();
int inputTokens = usage.inputTokenCount();
int outputTokens = usage.outputTokenCount();
```


29

FinishReason

- Enumération du package dev.langchain4j.model.output
- Indique pourquoi la génération du LLM s'est arrêtée
- `STOP` : le LLM a décidé qu'il avait fini de traiter la requête
- `LENGTH` : limite atteinte pour le nombre de token
- `TOOL_EXECUTION` : signale qu'il faut appeler un outil
- `CONTENT_FILTER` : arrêt à cause d'un contenu jugé contraire à la politique de modération du LLM
- `OTHER` : autre raison

30

Interface ChatLanguageModel (2/2)

- Méthodes par défaut de base :
  - `default String generate(String userMessage)` : ne reçoit que la réponse générée par le modèle, basée sur une question
  - `default Response<AiMessage> generate(ChatMessage… messages)` : reçoit une réponse générée par le modèle, basée sur une suite de messages de type System, User, AI

31

Exemples de messages

```
UserMessage userMessage1 =
    UserMessage.from("Hello, mon nom est Julien");
AiMessage aiMessage1 =
    model.generate(userMessage1).content();
// Bonjour Julien, comment puis-je vous aider ?
UserMessage userMessage2 =
    UserMessage.from("Quel est mon nom ?");
AiMessage aiMessage2 =
    model.generate(userMessage1, aiMessage1,
                   userMessage2).content();
// Julien
```


32

Classe d'un modèle concret

- LangChain4j contient une API pour chacun des LLMs qu'il supporte
- En particulier, cette API implémente l'interface `ChatLanguageModel`
- Par exemple, l'interface est implémentée par `GoogleAiGeminiChatModel` pour Gemini et `OpenAiChatModel` pour OpenAI
- Le plus souvent une instance de la classe d'implémentation s'obtient avec le pattern « builder », mais il peut y avoir d'autres moyens (pattern fabrique en particulier)

33

Exemple avec Gemini

- Avec un builder pour créer le modèle (décommenter si on veut des réponses au format JSON) :

```
ChatLanguageModel model =
    GoogleAiGeminiChatModel.builder()
      .apiKey(cle)
      .modelName("gemini-1.5-flash")
      .temperature(0.5)
      .timeout(Duration.ofSeconds(60))
//      .responseFormat(ResponseFormat.JSON)
      .build();
```

34

Exemple avec OpenAI

- Avec un builder pour créer le modèle :

```
ChatLanguageModel model =
    OpenAiChatModel.builder()
      .apiKey(cle)
      .modelName("gpt-4o-mini")
      .temperature(0.5)
//      .responseFormat("json_schema")
//      .strictJsonSchema(true)
      .build();
```

- Fabrique (avec une méthode `static` de la classe) :

```
ChatLanguageModel model =
    OpenAiChatModel.withApiKey(cle);
```


35

Paramètres de modèle

- `modelName` (`String`)
- `temperature` (`Double`)
- `max_tokens` (`Integer`)
- `frequencyPenalty` (`Double` entre -2.0 et 2.0 ; pénalise les tokens déjà utilisés)
- …

36

Exemple simple

```
String cle = System.getenv("GEMINI_KEY");
ChatLanguageModel modele = GoogleAiGeminiChatModel
                           .builder()
                           .apiKey(cle)
                           .modelName("gemini-1.5-flash")
                           .build();
String reponse = modele.generate("Capitale de la France ?");
System.out.println(reponse);
```


`PromptTemplate` (1/2)

- Package dev.langchain4j.model.input
- Pour obtenir des bonnes réponses à un LLM, il faut poser les questions avec un bon format ; un template permet d'imposer un format prédéfini
- Le constructeur prend en paramètre une `String` qui peut contenir des variables (par exemple {{nom}})
- Des variables sont prédéfinies :
  - `current_date` (`LocalDate.now()`)
  - `current_time` (`LocalTime.now()`)
  - `current_date_time` (`LocalDateTime.now()`)

Méthodes de `PromptTemplate`

- `static PromptTemplate from(String template)` : création d'un template
- `Prompt apply(Object valeur)` : appliquer une valeur pour un template qui n'a qu'une seule variable qui a nécessairement le nom {{it}}
- `Prompt apply(Map<String, Object> valeurs)` : appliquer des valeurs pour un template qui a plusieurs variables

Exemple avec variable prédéfinie

- Utilisation d'une variable prédéfinie :

```
Prompt prompt = PromptTemplate
    .from("Quel âge a-t-il en date du {{current_date}}?")
    .apply(Map.of());
reponse = modele.generate(prompt.text());
```


Exemple avec variable

- Les valeurs des variables non prédéfinies sont passés dans un objet (si une seule variable) ou une `Map`

```
Prompt prompt = PromptTemplate
    .from("Quel âge a {{it}} en date du {{current_date}}?")
    .apply("Quentin Tarantino");
```

```
Prompt prompt = PromptTemplate
    .from("Quel âge a {{name}} en date du {{current_date}}?")
    .apply(Map.of("name","Quentin Tarantino"));
```


Requête few shot avec template

```
PromptTemplate promptTemplate =
    PromptTemplate.from("""
Analyse le sentiment du texte qui suit. Réponds avec un
seul mot qui décrit le sentiment. Voici des exemples :
            INPUT: C'est une bonne nouvelle !
            OUTPUT: POSITIF

            INPUT: Pi est à peu près égal à 3.14
            OUTPUT: NEUTRE

            INPUT: Cette pizza n'est pas bonne !
            OUTPUT: NEGATIF

            INPUT: {{it}}
            OUTPUT: """);
```

@StructuredPrompt

- *Annotation* pour un type (classe)
- Simplifie la création de prompts complexes en les définissant dans une classe à part, en fonction des champs de la classe
- Attributs :
  - `String[]` value : template de prompt défini en une ou plusieurs lignes
  - `String` delimiter : si le template est défini en plusieurs lignes, elles seront jointes avec ce délimiteur

43

Exemple 1

```
@StructuredPrompt("Créer une recette de {{plat}} qui peut
être préparé en utilisant seulement {{ingredients}}")
public class PromptPourRecette {
  private String plat;
  private List<String> ingredients;
}
// Méthode main qui utilise CreateRecipePrompt :
PromptPourRecette.plat = "salade";
PromptPourRecette.ingredients = asList(
      "concombre", "tomate", "feta", "oignon", "olives");
Prompt prompt = StructuredPromptProcessor
                .toPrompt(PromptPourRecette);
AiMessage aiMessage =
    model.generate(prompt.toUserMessage()).content();
System.out.println(aiMessage.text());
```


44

Exemple 2 (plusieurs lignes) (1/2)

```
@StructuredPrompt({
   "Créer une recette de {{plat}} qui peut être préparé en
utilisant seulement {{ingredients}}.",
   "Structure ta réponse de cette façon :",
   "Nom de la recette : ...",
   "Description : ...",
   "Temps de preparation : ...",

   "Ingredients :",
   "- ...",
   "- ...",

   "Instructions:",
   "- ...",
   "- ..."
})
```


45

Exemple 2 (plusieurs lignes) (2/2)

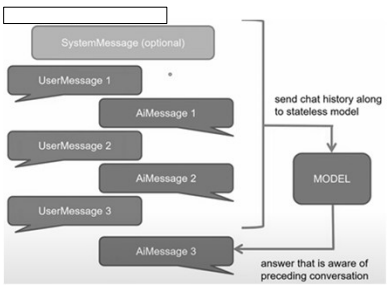
```
Prompt prompt =
    StructuredPromptProcessor.toPrompt(PromptPourRecette);

AiMessage aiMessage =
    model.generate(prompt.toUserMessage()).content();
```


46

Chat

- Les LLMs ne gardent pas l'état d'une conversation et l'application chat doit donc garder cet état



47

Interface ChatMemory

- Package dev.langchain4j.memory
- Représente l'état de la conversation durant un chat ; garde les derniers messages échangés avec le LLM
- Cet état peut être rendu persistant avec `ChatMemoryStore`
- Le développeur peut gérer lui-même cette mémoire :
  - Il peut mettre ce qu'il veut dans la mémoire, pas forcément un message réellement échangé avec le LLM
  - Il peut supprimer des messages de la mémoire pour alléger le coût, ou à cause du nombre limité de tokens qui peuvent être traités par un modèle

48

Méthodes de ChatMemory

- `void add(ChatMessage message)` : ajouter un message
- `List<ChatMessage> messages()` : liste des messages
- `void clear()` : supprime tous les messages
- `Object id()` : id de la chatMemory

49

Classes d'implémentation

- 2 classes implémentent l'interface `ChatMemory` :
  - `MessageWindowChatMemory` : ne retient que les N derniers messages
  - `TokenWindowChatMemory` : ne retient que les N derniers tokens
- Un seul `SystemMessage` ; si un nouveau message système est ajouté, l'ancien est supprimé
- Si le quota est atteint, le plus ancien item (message ou token) est supprimé de la mémoire
- La mémoire est conservée dans `ChatMemoryStore` ; par défaut `InMemoryChatMemoryStore` est utilisé

50

Création d'une mémoire

- Les 2 classes fournies utilisent le pattern « builder »
- Prenons le cas de `MessageWindowChatMemory`
- La classe contient la classe interne `static MessageWindowChatMemory.builder` dont une instance est renvoyée par la méthode `builder()`
- Le builder permet d'indiquer le type de `ChatMemoryStore` à utiliser pour conserver les messages et à fixer le nombre maximum de messages à conserver (il existe aussi un raccourci pour ce nombre dans la classe `MessageWindowChatMemory`)

51

Exemple

```
ChatMemory memory =
    MessageWindowChatMemory.withMaxMessages(10);
String question = "Films dirigés par Spielberg ?";
memory.add(new UserMessage(question));
Response<AiMessage> reponse =
    modele.generate(memory.messages());
memory.add(reponse.content()); // AiMessage ajouté

question = "Quel âge a-t-il ?";
memory.add(new UserMessage(question));
reponse = modele.generate(memory.messages());
System.out.println(reponse.content().text());
```

On verra qu'avec les Ai Services, la mémoire est gérée automatiquement

52

Interface ChatMemoryStore

- Implémentée par plusieurs classes dont `InMemoryChatMemoryStore`, `RedisChatMemoryStore`, `CassandraChatMemoryStore`
- Possible de récupérer des messages, de les supprimer ou de les modifier

53

Memory pour chaque utilisateur

```
interface Assistant {
  String chat(@MemoryId int memoryId,
              @UserMessage String userMessage);
}
Assistant assistant = AiServices.builder(Assistant.class)
   .chatLanguageModel(OpenAiChatModel.withApiKey(...))
   .chatMemoryProvider(memoryId ->
                       MessageWindowChatMemory.withMaxMessages(10))
   .build();
System.out.println(assistant.chat(1, "Hello, my name is Klaus"));
        // Hi Klaus! How can I assist you today?
System.out.println(assistant.chat(2, "Hello, my name is Francine"));
        // Hello Francine! How can I assist you today?
System.out.println(assistant.chat(1, "What is my name?"));
        // Your name is Klaus.
System.out.println(assistant.chat(2, "What is my name?"));
        // Your name is Francine.
```


54

Utilisation du logging

- Méthodes des builders des modèles pour indiquer si on veut activer le logging sur les interactions avec le modèle
- Dépend du modèle utilisé ; pour `OpenAIChatModel`
  - `logRequests(boolean)` :
  - `logResponse(boolean)` :
- Pour `GoogleAiGeminiChatModel`
  - `logRequestsAndResponses(boolean)`

55

Exemple de logging

```
ChatLanguageModel model = GoogleAiGeminiChatModel.builder()
    .apiKey(openAiKey)
    .logRequestsAndResponses(true)
    .build();
```


56

Configuration logging

- Pour que le logging fonctionne, il faut le configurer
- LangChain4j utilise SLF4J (Simple Logging Facade for Java ; https://www.slf4j.org/), façade pour de nombreux frameworks de logging
- Il faut choisir un des frameworks de logging supportés par SLF4J ; par exemple, celui du JDK (java.util.logging, JUL), Logback ou Log4j
- Pour configurer le logging, il faut configurer le système de logging sous-jacent

57

Maven pour logging

- Il faut ajouter une dépendance vers le framework utilisé ; par exemple, si on utilise java.util.logging :

```
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-jdk14</artifactId>
  <version>2.0.16</version>
</dependency>
```


58

Configuration par code pour JUL

```
import java.util.logging.Logger;
...
private static void configureLogger() {
  // Configure le logger sous-jacent (java.util.logging)
  Logger packageLogger =
      Logger.getLogger("dev.langchain4j");
  packageLogger.setLevel(Level.FINE); // Ajuster niveau
  // Ajouter un handler pour la console pour faire
  // afficher les logs
  ConsoleHandler handler = new ConsoleHandler();
  handler.setLevel(Level.FINE);
  packageLogger.addHandler(handler);
}
```


59

Parser de output

- Permet de convertir les réponses générées par un LLM en types de données structurés, facilitant leur traitement par l'application
- Particulièrement utiles pour garantir que les réponses suivent un format attendu, comme un objet JSON, une liste ou une classe spécifique
- Utiliser les AI services étudiés plus loin

60

Modèles d'embeddings

- Utiliser les classes des modèles concrets qui ont des modèles d'embeddings
- Il faut choisir le modèle qui convient le mieux à ce que l'on veut :
  - A quoi vont servir les embeddings
  - Coûts
  - Rapidité
  - Efficacité

61

Exemple modèle d'embeddings

```
String llmKey = System.getenv("GEMINI_KEY");
EmbeddingModel modele = new GoogleAiEmbeddingModel(
        "text-embedding-004", llmKey, 2,
        TaskType.SEMANTIC_SIMILARITY, "",
        200, Duration.ofSeconds(10), true);
String phrase1 = "Bonjour, comment allez-vous ?";
String phrase2 = "Salut, quoi de neuf ?";
Response<Embedding> reponse1 = modele.embed(phrase1);
Response<Embedding> reponse2 = modele.embed(phrase2);
Embedding emb1 = reponse1.content();
Embedding emb2 = reponse2.content();
// Calcul de similarité cosinus entre les 2 embeddings
double similarity = CosineSimilarity.between(emb1, emb2);
System.out.println("Similarité cosinus : " + similarity);
```


62

AiServices

63

Chaînes et services IA

- Pour créer des flux automatisés et structurés autour des LLMs
- Chaînes : inspirées de LangChain ; permettent de combiner plusieurs étapes d'utilisation des LLMs ; ne sont pas étudiées dans ce support
- Services IA (AiServices) : autre solution que les chaînes, mieux adapté à Java que les chaînes

64

Service IA

- Définit un comportement pour des échanges de messages entre l'application et le LLM
- Le développeur définit une interface Java qui contient les méthodes qui correspondent aux interactions avec le LLM
- LangChain4j implémente les méthodes de l'interface ; les échanges de messages (questions et réponses) entre l'application et le LLM sont implémentées
- LangChain4j tient compte des types des paramètres, du type retour et des annotations de chaque méthode pour écrire son implémentation

65

Exemple d'interface

```
public interface Assistant {
    String chat(String prompt);
}
```

- L'application pourra appeler la méthode `chat` en lui passant une `String` en paramètre
- Un message sera alors envoyé au LLM
- La réponse du LLM sera retournée par la méthode `chat`
- Pour que tout cela fonctionne, il faut créer un assistant avec la classe `AiServices` : `AiServices.builder(MonAssistant.class)`

66

Classe AiServices<T>

- Package dev.langchain4j.service
- Cette classe crée un service IA en implémentant une interface définie par le développeur (création avec méthode `create` pour les cas simples, ou bien avec un builder)
- `T` est l'interface pour laquelle `AiServices` va fournir une implémentation
- Pas de memory par défaut

67

Supporté par AiServices

- Presque tout ce qui est fait avec l'API de bas niveau de LangChain4j peut être fait avec les AI services :
  - Mémoire pour la conversation
  - RAG (`RetrievalAugmentor` et `ContentRetriever`)
  - Outils (Tools)
  - Streaming
  - Auto-modération
  - ...

68

Méthodes de AiServices<T> (1/4)

- `public static <T> T create(Class<T> aiService, ChatLanguageModel chatLanguageModel)` : crée simplement un service IA pour le modèle indiqué ; variante avec `StreamingChatLanguageModel` pour le 2ème paramètre
- `public static <T> AiServices<T> builder(Class<T> aiService)` : pour création plus complexe
- `public abstract T build()` : construit et retourne le service IA

69

Méthodes de AiServices<T> (2/4)

- `public AiServices<T> chatLanguageModel(ChatLanguageModel model)` : indique le modèle qui sera utilisé par le service IA
- `public AiServices<T> streamingChatLanguageModel( StreamingChatLanguageModel model)` : variante pour streaming
- `public AiServices<T> chatMemory(ChatMemory chatMemory)` : indique la mémoire qui sera utilisée par le service IA (pas de mémoire par défaut) ; si on veut une mémoire différente pour chaque user, utiliser la méthode `chatMemoryProvider` qui prend en paramètre un `ChatMemoryProvider` (voir exemple à suivre)

70

Exemple chatMemoryProvider

```
interface Assistant {
   String chat(@MemoryId int memoryId,
               @UserMessage String message);
 }

Assistant assistant = AiServices.builder(Assistant.class)
   .chatLanguageModel(OpenAiChatModel.withApiKey(...))
   .chatMemoryProvider(
       memoryId -> MessageWindowChatMemory
                   .withMaxMessages(10))
   .build();
```


71

Méthodes de AiServices<T> (3/4)

- `public AiServices<T> tools(List<Object> objectsWithTools)` : configure les outils que le LLM pourra utilizer ; une mémoire d'au moins 3 messages est requise
- `public AiServices<T> contentRetriever(ContentRetriever contentRetriever)` : configure un retriever qui sera appelé pour chaque exécution de méthode du service IA pour retrouver le contenu associé à un message utilisateur depuis une source de données (par exemple un magasin d'embeddings dans le cas d'un `EmbeddingStoreContentRetriever`)

72

Méthodes de `AiServices<T>` (4/4)

- `public AiServices<T> retrievalAugmentor(RetrievalAugmentor retrievalAugmentor)` : configure un `RetrievalAugmentor` qui sera appelé pour chaque exécution de méthode du service IA ; un `RetrievalAugmentor` ajoute un `UserMessage` avec un contenu retrouvé par un retriever ; on peut utiliser le `DefaultRetrievalAugmentor` fourni par LangChain4j ou en implémenter un autre
- 3 autres méthodes liées à la modération

Exemple simple

```
Assistant assistant = AiServices.builder(Assistant.class)
                      .chatLanguageModel(modele)
String reponse = assistant.chat("Hello, world");
System.out.println(reponse);
```


Exemple simple pour chat

```
ChatLanguageModel modele = GoogleAiGeminiChatModel.builder()
                           .apiKey(cle)
                           .modelName("gemini-1.5-flash")
                           .build();
ChatMemory chatMemory =
    MessageWindowChatMemory.withMaxMessages(10);
Assistant assistant = AiServices.builder(Assistant.class)
                      .chatLanguageModel(model)
                      .chatMemory(chatMemory)
                      .build();
String rep1 = assistant.chat("Hello! je m'appelle Julie");
System.out.println(rep1); // Hello Julie ! Comment ... ?
String rep2 = assistant.chat("Quel est mon nom?");
System.out.println(rep2); // Votre nom est Julie.
```


Comment ça marche ?

- `AiServices` crée un objet proxy qui implémente l'interface
- La réflexivité est utilisée pour cela
- Le proxy s'occupe, entre autres,
  - d'envoyer les requêtes au LLM dans un format compatible avec le LLM (souvent JSON)
  - de recevoir les réponses du LLM (souvent JSON)
  - de gérer les erreurs et exception
  - de convertir les paramètres et les valeur retour des méthodes de l'interface ; par exemple pour transformer une `String` en `UserMessage`, en entrée, et un `AiMessage` en `String`, en sortie

Types retour méthodes interface

- `String`, `AiMessage` ou `Response<AiMessage>` si on veut la réponse du LLM tel quel
- De nombreux autres types sont possibles :
  - `List<String>` ou `Set<String>`
  - `Map<K,V>`
  - Une énumération ou un `boolean` (par exemple si on veut utiliser le LLM pour une classification)
  - Un type primitif, une classe qui enveloppe un type primitif, `Date`, `LocalDateTime`, `BigDecimal`, ...
  - Ou même une classe quelconque (POJO) ; voir « Extraction de données » plus loin

Annotations méthodes

- On peut indiquer des messages système ou utilisateur qui feront partie de la conversation au moment où l'interaction associée à la méthode commencera
- Ces messages peuvent être des templates et donc contenir des variables ; les valeurs des variables sont définies par les paramètres des méthodes de l'interface (voir `@V`)
- Si une méthode n'a qu'un seul paramètre, la variable spéciale « `it` » prend la valeur de l'argument passé en paramètre (voir section « Extraction de données »)

Annotation @V

- Annote un paramètre d'une méthode d'une interface qui définit un service IA
- L'attribut `value` de cette annotation correspond au nom d'une variable d'un template d'un message qui annote la même méthode
- Quand la méthode de l'interface est appelée, les valeurs des paramètres sont appliquées au template, ce qui envoie les messages correspondant au modèle

Exemple

```
interface Traducteur {
  @SystemMessage("""
    Tu es un traducteur professionnel de français en
    {{langue}}""")
  @UserMessage("Traduis le texte suivant: {{texte}}")
  String traduire(@V("texte") String texte,
                  @V("langue") String langue);
 }

Traducteur traducteur = AiServices.builder(Traducteur.class)
                        .chatLanguageModel(modele).build();

String italien = traducteur.traduire("Hello", "italien");
```


Extraction de données

Pour les types standard supportés

- Il est souvent intéressant de récupérer une information structurée à partir d'un texte non structuré
- On peut demander d'extraire un des types retour supportés par LangChain4j pour les méthodes des AI services
- L'exemple suivant montre comment récupérer une date (`LocalDate`) ou un temps (`LocalTime`) d'un texte

DateTimeExtractor (1/2)

```
public interface ExtracteurDateTemps {
  @UserMessage("Extrait la date de {{it}}")
  LocalDate extraireDate(String texte);

  @UserMessage("Extrait le temps de {{it}}")
  LocalTime extraireTemps(String texte);

  @UserMessage("Extrait la date et le temps de {{it}}")
  LocalDateTime extraireDateEtTemps(String texte);
}
```


DateTimeExtractor (2/2)

```
ExtracteurDateTemps extracteur =
    AiServices.builder(ExtracteurDateTemps.class)
              .chatLanguageModel(model)
              .build();
String texte = """
  La tranquillité régnait dans la soirée de 1968, à un
  quart d'heure de minuit, le jour après Noël.""");

LocalDate date = extracteur.extraireDate(texte);
LocalTime temps = extracteur.extraireHeure(texte);
// Que sera-t-il affiché ?
System.out.println(date + "; " + temps);
```

Pour les classes Java

- L'extraction fonctionne aussi sous la forme de classes ou de records Java, créés dans l'application
- Par exemple, dans le texte « Christophe Colomb, né en 1451 sur le territoire de la république de Gênes et mort le 20 mai 1506 à Valladolid, est un navigateur génois au service des Rois catholiques, Isabelle de Castille et Ferdinand d'Aragon. », il peut être intéressant de récupérer dans une classe Java, le nom, les dates et lieux de naissance et de mort du personnage dont on parle
- Voyons comment faire

85

Etapes

1. Créer le modèle **en demandant une réponse au format JSON**
2. Ecrire l'interface de l'extracteur d'information, un AI service
3. Ecrire la structure de l'information à récupérer sous la forme d'une classe ou d'un record Java, **en utilisant des noms significatifs**
4. Lancer l'exécution de l'extracteur

86

Exemple, étape 1

```
ChatLanguageModel modele =
    GoogleAiGeminiChatModel
    .builder()
    .apiKey(llmKey)
    .modelName("gemini-1.5-flash")
    .responseFormat(ResponseFormat.JSON)
    .build();
```


87

Exemple, étape 2

```
public interface ExtracteurInfosPersonne {
  @UserMessage("""
    Extrait les informations sur la personne du texte
    ci-dessous :
    ---
    {{it}}
    ---
    """)
    Personne extraireInfosPersonne(String texte);
}
```


88

Exemple, étape 3

```
public record Personne(String nom,
                       LocalDate anNaissance,
                       String lieuNaissance) { }
```


89

Exemple, étape 4

```
ExtracteurInfosPersonne extracteur =
    AiServices.builder(ExtracteurInfosPersonne.class)
              .chatLanguageModel(modele).build();
Personne personne = extracteur.extraireInfosPersonne (
    """
    Christophe Colomb, né en 1451 sur le territoire de la république de
    Gênes et mort le 20 mai 1506 à Valladolid, est un navigateur génois
    au service des Rois catholiques, Isabelle de Castille et Ferdinand
    d'Aragon.
    """);
System.out.println(personne.nom());
System.out.println(personne.anNaissance());
```


90

Description pour extraction

- Si le type retour est un type Java et si les contenus des attributs ne sont pas clairs d'après leur nom, on peut ajouter l'annotation `@Description` qui précise les choses
- Par exemple

```
public record Personne(
    String nom,
    @Description("Année de naissance") int anNaiss,
    String lieuNaissance) {
}
```


91

Outils

92

Appel de fonction des LLMs

- Les LLMs ont des lacunes ; par exemple ils ne sont pas bons en mathématiques et ils ne peuvent consulter Internet en temps réel
- Pour combler ces lacunes, de nombreux LLMs, dont OpenAI et Gemini, ont ajouté la possibilité de définir des fonctions personnalisées qui seront exécutées dans le processus de génération

93

Etapes (1/2)

1. Ecrire les fonctions dans un langage informatique
2. Préparer les descriptions des fonctions en JSON : nom de la fonction, description de ce qu'elle fait en langage naturel, liste des paramètres avec leur type et une description
   Chaque description sera utilisée par le LLM pour savoir si la fonction lui sera utile pour répondre à la question
3. Envoyer une 1ère requête au LLM, avec une question et des descriptions de fonctions ; il répond en indiquant quelles fonctions il utilisera pour répondre à la question ; la réponse contiendra un champ « tool_calls »

94

Exemple de réponse avec tool_calls

```
{   finish_reason: 'tool_calls',
    index: 0,
    logprobs: null,
    message: {
        content: null,
        role: 'assistant',
        function_call: null,
        tool_calls: [
            {
                id: 'call_62136354',
                function: {
                    arguments: '{"commande_id":"12345"}',
                    name: 'dateLivraison'
                },
                type: 'function'
            }
        ]
    } }
```


95

Etapes (2/2)

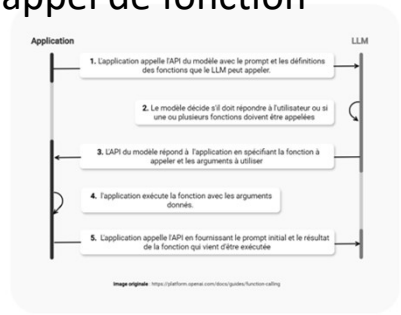
4. De la réponse du LLM à la 1ère requête, extraire les noms des fonctions avec les valeurs de leurs paramètres
5. Exécuter les fonctions avec leurs paramètres dans le programme qui utilise l'API du LLM
6. Envoyer dans une 2ème requête au LLM, les résultats de l'exécution des fonctions dans un message dont le rôle est « tool » ; ce résultat sera utilisé par le LLM pour donner sa réponse finale à la question

96

Exemple 2ème requête

```
{
   "role": "tool",
  {
    "order_id":"12345",
     "dateLivraison": "<date retournée par fonction>"
  },
  "tool_call_id": "call_62136354"
}
```


Diagramme de séquence pour un appel de fonction



Apport de LangChain4j

- Le processus est complexe :
  1. Il faut envoyer 2 requêtes
  2. Extraire des informations de la 1ère requête
  3. Utiliser ces informations pour appeler la fonction
  4. Donner le résultat de l'exécution dans la 2ème requête
- LangChain4j simplifie le processus :
  - La fonction (méthode Java) est annotée par `@Tool` qui décrit la fonction en langage naturel
  - Tout le reste est automatisé par LangChain4j

Exemple - la question

- Question : quelle est la racine carrée de la somme des nombres des lettres dans les mots « bonjour » et « Monsieur » ? (la réponse est √14)

Exemple - le code

```
Assistant assistant = AiServices.builder(Assistant.class)
    .chatLanguageModel(LlmChatModel.withApiKey(…))
    .tools(new Calculator())
    .chatMemory(MessageWindowChatMemory.withMaxMessages(10))
    .build();
String question = "Quelle est la racine carrée des nombres de
lettres dans les mots 'bonjour' et 'Monsieur' ? ;
String answer= assistant.chat(question);
```

```
interface Assistant {
  String chat(String userMessage);
}
```


Exemple - le code des outils

```
class Calculator {
  @Tool("Calcule la longueur d'une chaîne de caractères")
  int stringLength(String s) {
    return s.length();
  }
  @Tool("Calcule la somme de deux entiers")
  int add(int a, int b) {
    return a + b;
  }
  @Tool("Calcule la racine carrée d'un entier")
  double sqrt(int x) {
    return Math.sqrt(x);
  }
}
```

Exemple avec devises (1/2)

```
@Tools("Convertit un montant monétaire de la devise
{{de}} vers la devise {{vers}}")
public double convert(double montant, @V("de") String de,
@V("vers") String vers) {
  return montant * taux.get(de) / taux.get(vers);
}

@Tool("Obtient le nom d'une devise à partir de son
abréviation")
public String getCurrency(String abreviation) {
  return devises.get(abreviation);
}
```


103

Exemple avec devises (2/2)

```
String question = """
   Sur le site Vente.com de plusieurs pays, un ordinateur
   coûte 718.25 GPB, 83,900 INR, 749.99 USD,
   and 177,980 JPY. Quelle est la meilleure offre ?
   Dans le résultat, écris le nom de la devise et le
   montant dans cette devise.
""";
String reponse = assistant.chat(question);
```


104

@P, @Description

- Ces 2 annotations permettent de donner une description en langage naturel
  - d'un paramètre d'une méthode d'outil (`@P`)
  - d'une classe d'outils, d'un champ d'une telle classe (`@Description`)

  pour aider le LLM à choisir les outils qu'il va utiliser
- En plus de l'attribut `value`, l'annotation `@P` a un attribut `required` pour indiquer si le paramètre est requis

105

Exemple @Description

```
@Description("Requête à exécuter")
class Query {

  @Description("Les champs à sélectionner")
  private List<String> select;

  @Description("Conditions de filtrage")
  private List<Condition> where;
}

@Tool
Result executeQuery(Query query) {
  ...
}
```


106

Exemple @P

```
class OutilMétéo {
    @Tool("Renvoie les prévisions météo pour une ville")
    String previsions(
        @P(value="Ville dont on veut les prévisions météo",
           required=true)
        String ville,
        TemperatureUnit temperatureUnit) {
        ...
    }
}
```


107

Modération

108

Présentation

- Dans une application d'entreprise il est important de s'assurer que les contenus échangés avec les clients et partenaires de l'entreprise sont appropriés et sûrs
- L'entreprise ne doit pas injurier ses clients et elle ne doit pas répondre à des propos dangereux, injurieux, racistes ou sexistes

@Moderate

- Package dev.langchain4j.service
- Il est très simple de « modérer » automatiquement les conversations avec `AiServices` ; il suffit d'ajouter l'annotation `@Moderate` sur les méthodes de l'assistant
- Quand une méthode est annotée avec cette annotation, chaque appel de la méthode déclenche un appel en parallèle au modèle de modération
- Avant la fin de la méthode, le résultat du modèle de modération est attendu
- Si le modèle de modération signale un problème, une `ModerationException` est lancée par la méthode

Comment modérer

1. Créer un `ModerationModel`
2. A la création de l'assistant par `AiServices`, indiquer que l'on veut modérer les conversations avec ce `ModerationModel`
3. Il suffit ensuite d'ajouter l'annotation `@Moderate` sur les méthodes de l'assistant

Interface `ModerationModel`

- Package dev.langchain4j.model.moderation
- Implémenté par `OpenAiModerationModel`, `MistralAiModerationModel`, `DisabledModerationModel` (uniquement pour tests)
- 5 méthodes `moderate` surchargées pour modérer un texte, ou un ou plusieurs messages de l'utilisateur

Classe `OpenAiModerationModel`

- Builder ou construction avec le nom du modèle qui est une des valeurs de l'énumération `OpenAiModerationModelName` :
  - `TEXT_MODERATION_LATEST`
  - `TEXT_MODERATION_STABLE`

Exemple (1/3)

```
interface Chat {
  @Moderate
  String chat(String text);
}
```

Exemple (2/3)

```
OpenAiModerationModel moderationModel =
    OpenAiModerationModel.builder()
        .apiKey(cleOpenAi)
        .modelName(TEXT_MODERATION_LATEST)
        .build();

ChatLanguageModel chatModel = ...;

Chat chat = AiServices.builder(Chat.class)
            .chatLanguageModel(chatModel)
            .moderationModel(moderationModel)
            .build();
```


115

Exemple (3/3)

```
try {
  chat.chat("Je vais tuer tout le monde !!!");
} catch (ModerationException e) {
  System.err.println(e.getMessage());
  // "Je vais tuer tout le monde !!!" violates content
  // policy
}
```


116

Streaming

117

Introduction

- Les modèles génèrent leurs réponses token par token
- Il est possible de récupérer les tokens dès qu'ils sont générés, plutôt que d'attendre que le modèle ait généré toute sa réponse
- Pour cela, il faut utiliser l'interface `StreamingChatLanguageModel`, à la place de `ChatLanguageModel`
- Le développeur doit implémenter un handler pour indiquer ce qui se passera quand l'application recevra les tokens

118

`StreamingChatLanguageModel`

- Package dev.langchain4j.model..chat
- 5 méthodes `generate` qui retournent `void` et ont un `StreamingResponseHandler<AiMessage>` en dernier paramètre
- Le 1er paramètre représente le ou les messages de la conversation

119

Streaming avec AI services

- Les AI services permettent de faire simplement du streaming
- Utiliser `TokenStream` comme type retour des méthodes de l'interface

120

Exemple de AI service avec streaming et mémoire (1/2)

```
interface Assistant {
  TokenStream chat(String message);
}

// Création d'un assistant :
Assistant assistant = AiServices.builder(Assistant.class)
     .streamingChatLanguageModel(modele)
     .chatMemory(chatMemory)
     .build();
```


121

Exemple de AI service avec streaming et mémoire (2/2)

```
TokenStream tokenStream = assistant.chat("...");
tokenStream
  .onNext(token -> System.out.println(token))
  .onComplete(
      response ->
          System.out.println("Streaming terminé "
                "avec la réponse + response.content.text));
  .onError(error -> error.printStackTrace())
  .start();
```


122

Interface `TokenStream`

- Package dev.langchain4j.service
- Représente un stream de tokens
- La méthode `start()` démarre l'envoi de la requête au LLM et le traitement des tokens émis par le LLM en réponse
- On peut recevoir des notifications
  - quand un nouveau token est disponible (méthode `onNext` prend en paramètre comment sera utilisé le token
  - quand le LLM a terminé sa réponse (`onComplete`)
  - quand une erreur est survenue pendant le streaming (`onError`)

123

Streaming avec JSF

- Comment faire pour que les tokens envoyés par le LM soient affichés immédiatement et automatiquement sur une page JSF de l'interface utilisateur ?
- 2 solutions :
  - Faire du polling avec JavaScript (le code JavaScript peut sonder à intervalles réguliers le serveur pour savoir si des nouveaux tokens ont été générés)
  - Utiliser un WebSocket

124

APIs et outils autour de IA

125

LlamaIndex

- Comme LangChain, facilite l'intégration des LLMs dans des applications, mais avec des approches différentes
- LlamaIndex
  - fournit des outils pour structurer et indexer des données non ou semi-structurées (documents PDF, BDs, APIs,...)
  - s'intègre au RAG pour connecter des LLMs à des sources de données externes (avec embeddings ou autres techniques)
  - permet de créer des structures d'index personnalisées pour optimiser la récupération des informations, et d'interroger l'index avec un langage naturel.

126

LangSmith (1/2)

- Plateforme de développement complète pour aider à passer de la phase de prototype à celle de production avec des applications basées sur des LLMs
- Offre un ensemble d'outils pour optimiser chaque étape du projet
- Pour débogage, collaboration avec une équipe de développement, tests et surveillance des applications

127

LangSmith (2/2)

- Principales fonctionnalités :
  - Suivi de traces (prompts, réponses LLM, contextes)
  - Intégration avec LangChain (version Java en développement)
  - Expérimentation de différentes configurations de modèles et de prompts pour optimiser les résultats

128

- Whisper de OpenAI pour « speach to text ». Librairie Java-Whisper. Vidéo YouTube sur cette librairie : https://www.youtube.com/watch?v=ZeH3bBKdqRU. Cette vidéo s'appuie sur le tutoriel de OpenAI https://platform.openai.com/docs/tutorials/meeting-minutes.
- Gradio pour créer des interfaces utilisateur Web pour les modèles de machine learning et de les déployer sans écrire de code??**??

129

- Extraire la transcription/sous-titres de ce qui est dit dans une vidéo YouTube en utilisant l'API YouTube :
  - https://developers.google.com/youtube?hl=fr pour gérer les vidéos YouTube ; plus particulièrement https://developers.google.com/youtube/v3/docs/captions?hl=fr pour travailler avec les transcriptions
  - Autres possibilités pour travailler avec les transcriptions :
    - Librairie youtube-transcript-api (seulement pour Python) https://pypi.org/project/youtube-transcript-api/
    - Captions grabber https://www.captionsgrabber.com/ un site Web pour travailler avec les sous-titres ; voir vidéo de démonstration https://www.youtube.com/watch?v=OS54TX3YptE

130

Autres liens intéressants

- https://platform.openai.com/docs/tutorials/web-qa-embeddings. Comment parcourir un site Web pour transformer les pages en « embeddings »

131

- Projet GitHub https://github.com/kousen/openaidemo en Java 17 pour utiliser Whisper, la génération d'image avec PicoGen, DallE et Stable Diffusion ; voir https://www.youtube.com/watch?v=vRvlqFQGLzQ&list=PLZOgUaAUCiT7ooFAUWId7oeWatv6b30CD

132

ElevenLabs

- https://elevenlabs.io/
- Pour « Text to Speech »

133

Outils pour écrire du code

- https://github.com/jamesmurdza/awesome-ai-devtools, par James Murdza

134

Type d'outils pour coder

- Complétion de code : GitHub Copilot (gratuit pour universitaires ; https://github.com/features/copilot), Codeium (https://www.codium.ai/ ; version gratuite)
- Assistant pour coder : on peut chater avec un assistant ; vo pour HTML (https://vo.dev/chat)
- Générateur d'interface utilisateur
- Générateur d'applications
- Documentation
- Contrôle de version
- Aide pour les tests

135

Erreurs avec outils IA pour coder

- Référence de fichiers qui n'existent pas
- Code qui utilise d'anciennes versions des librairies
- Ne pas oublier de remplacer certaines parties du code généré
- Fichiers pas dans le bon chemin
- Erreurs de logique
- Mauvaise compréhension de ce que veut le développeur

136

Références

137

IA avec Java (1/2)

- « AI for Java developers » par Microsoft : https://www.youtube.com/watch?v=V45tKEYYAFs&list=PLPeZXlCR7ew8sdUWqf2itkRG5BUE7GFsy
- TensorFlow pour Java : https://www.baeldung.com/tensorflow-java https://www.tensorflow.org/jvm/install?hl=fr
- Tensorflow et Keras pour Java : https://github.com/dhruvrajan/tensorflow-keras-java Deeplearning4j, https://deeplearning4j.konduit.ai/
- LangChain4j, https://github.com/langchain4j/langchain4j

138

IA avec Java (2/2)

- Playlist YouTube sur IA en Java : https://www.youtube.com/playlist?list=PLZOgUaAUCiT700FAUWId70eWatv6b30CD

139

API d'OpenAI

- OpenAI : https://openai.com
- API de OpenAI : https://platform.openai.com/
- Documentation sur l'API : https://platform.openai.com/docs/api-reference/chat
- Guide pour utiliser l'API de complétion : https://platform.openai.com/docs/guides/text-generation/chat-completions-api

140

API de Gemini

- Documentation sur l'API : https://ai.google.dev/gemini-api/docs

141

LangChain

- https://www.langchain.com/
- Documentation : https://python.langchain.com/docs/get_started/introduction
- Modules : https://python.langchain.com/docs/how_to/#components
- Quelques articles sur LangChain :
  - https://www.lemondeinformatique.fr/actualites/lire-langchain-un-framework-qui-facilite-le-developpement-autour-des-llm-91921.html
  - https://www.ibm.com/fr-fr/topics/langchain
  - https://aws.amazon.com/fr/what-is/langchain/
  - https://www.lemagit.fr/conseil/Lessentiel-sur-LangChain

142

LangChain4j (1/4) - liens officiels

- https://langchain4j.github.io/langchain4j/
- Documentation : https://docs.langchain4j.dev/
- Javadoc : https://langchain4j.github.io/langchain4j/apidocs/index.html, https://docs.langchain4j.dev/apidocs/index.html
- Code source (attention, utilise Lombok et des getters et setters n'apparaissent ni dans le code, ni dans la javadoc) : https://github.com/langchain4j/langchain4j
- Tutoriels : https://langchain4j.github.io/langchain4j/tutorials/, https://docs.langchain4j.dev/category/tutorials
- Exemples : https://github.com/langchain4j/langchain4j-examples, https://github.com/langchain4j/langchain4j-examples/tree/main/other-examples/src/main/java

143

LangChain4j (2/4)

- https://www.youtube.com/watch?v=cjI_6Siry-s
- https://www.youtube.com/watch?v=Ewr1KYPtLao
- AI services : https://www.sivalabs.in/langchain4j-ai-services-tutorial/, https://docs.langchain4j.dev/tutorials/ai-services
- Generative AI Conversations using LangChain4j ChatMemory : https://www.sivalabs.in/generative-ai-conversations-using-langchain4j-chat-memory/
- Getting Started with Generative AI using Java, LangChain4j, OpenAI and Ollama : https://www.sivalabs.in/getting-started-with-generative-ai-using-java-langchain4j-openai-ollama/

144

LangChain4j (3/4)

- LangChain4j. Webinar organisé par Arun Gupta, avec Marcus Hellberg ; à la fin montre comment on peut charger un contexte personnalisé pour que le bot en tienne compte dans sa démo : https://twitter.com/i/broadcasts/1yNxaZyPzXRKj
- Avec Quarkus, passe un code HTML à l'API OpenAI et résume le contenu du HTML ; le code HTML est passé en morceaux : https://developers.redhat.com/articles/2024/02/07/how-use-llms-java-langchain4j-and-quarkus#exploring_the_capabilities_of_langchain4j_and_quarkus

145

LangChain4j (4/4)

- https://kindgeek.com/blog/post/experiments-with-langchain4j-or-java-way-to-llm-powered-applications : article très complet sur LangChain4j (6/2/24)
- Stackoverflow : https://stackoverflow.com/questions/tagged/langchain4j
- Discord : https://discord.com/channels/1156626270772269217/1156626271212666882

146

LangChain4j et Jakarta EE

- Projet « Smallrye LLM », intégré à Jakarta EE et MicroProfile : https://github.com/smallrye/smallrye-llm ; Smallrye est une implémentation de Eclipse MicroProfile (https://smallrye.io/)

147

MOOCs Udemy sur LangChain

- https://www.udemy.com/course/langchain-in-action-develop-llm-powered-applications/
- https://www.udemy.com/course/langchain-python-french/ ; bonne présentation gratuite (environ 1 heure), traduction automatique en français de Melt Labs d'un cours en anglais ; Python

148

MOOCs et vidéos sur LangChain

- La playlist de courtes vidéos sur les LLMs, ChatGPT, LangChain : https://www.youtube.com/playlist?list=PLKWoAUxdZEb_BqGgm-Rk7fiPUXccFHBGB
- https://www.youtube.com/watch?v=uJJ6uP5IViA
- https://www.youtube.com/watch?v=1VeYoNoXIgU

149

- Série de 4 courtes vidéos qui montrent comment utiliser Hugging Face et Ollama pour faire du fine-tuning avec Llama-3.2
  - https://www.youtube.com/watch?v=RAubwMSPRTo
  - https://www.youtube.com/watch?v=wco_8l_zh7s
  - https://www.youtube.com/watch?v=VePkG2EQKIM
- Semantic Kernel : kit de développement open source pour faciliter l'utilisation de l'IA en Java, Python, C# https://learn.microsoft.com/en-us/semantic-kernel/overview/

150