

LangChain4j

EMSI - Université Côte d'Azur
Richard Grin
Version 2.37 - 28/10/25

1

Plan du support

- LangChain
- Présentation et bases de LangChain4j
- AiServices
- Extraction des données
- Outils
- Modération
- Streaming
- APIs et outils autour de l'IA
- Références

R. Grin

LangChain4j

2

2

LangChain

R. Grin

LangChain4j

3

3

Présentation

- <https://www.langchain.com/>
- Framework open source pour développer des applications qui utilisent des LMs
- Couche abstraite pour travailler avec l'API de nombreux LMs (OpenAI, Gemini, Llama, Anthropic, Mistral, ...)
- Permet de combiner des LMs 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

R. Grin

LangChain4j

4

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

R. Grin

LangChain4j

5

5

Présentation et bases de LangChain4j

- Modèles
- Templates
- Mémoire
- Trace et logging
- Modèles embeddings

R. Grin

LangChain4j

6

6

Présentation

- <https://langchain4j.github.io/langchain4j/>
- Au départ, transposition de LangChain à Java, mais il s'en est progressivement différencié pour une intégration plus naturelle à Java
- Standard de facto pour IA avec Java

R. Grin

LangChain4j

7

7

Ce qui est offert

- APIs unifiées pour les LMs, les BD vectorielles (pour embeddings)
- Boîte à outils pour prompts, gestion de la mémoire, traitement des ressources externes, RAG, « outils » des LMs, agents
- Intégration avec les principaux frameworks Java (Jakarta EE, Spring Boot, Quarkus, Micronaut,...)
- Documentation très complète avec de nombreux exemples de code pour des cas d'utilisation

R. Grin

LangChain4j

8

8

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 embeddings, les magasins/entrepôts ...
 - Plus haut niveau en utilisant AiServices (configuration déclarative)

R. Grin

LangChain4j

9

9

Intégration avec les LMs

- Anthropic (texte, image)
- Azure OpenAI
- Google AI Gemini (avec texte, image, audio, vidéo, PDF)
- HuggingFace
- Jlama
- LocalAI
- Mistral AI (texte, image)
- Ollama
- OpenAI (avec texte, image, audio, PDF)
- ...

R. Grin

LangChain4j

10

10

Dépendances Maven - Gemini

```
<dependency>
  <groupId>dev.langchain4j</groupId>
  <artifactId>langchain4j</artifactId>
  <version>1.7.1</version>
</dependency>
<dependency>
  <groupId>dev.langchain4j</groupId>
  <artifactId>langchain4j-google-ai-gemini</artifactId>
  <version>1.7.1</version>
  <type>jar</type>
</dependency>
```



R. Grin

LangChain4j

11

11

Dépendances Maven - OpenAI

```
<dependency>
  <groupId>dev.langchain4j</groupId>
  <artifactId>langchain4j</artifactId>
  <version>1.7.1</version>
</dependency>
<dependency>
  <groupId>dev.langchain4j</groupId>
  <artifactId>langchain4j-open-ai</artifactId>
  <version>1.7.1</version>
</dependency>
```

R. Grin

LangChain4j

12

12



Clé secrète pour API

- La plupart des LMs 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 :

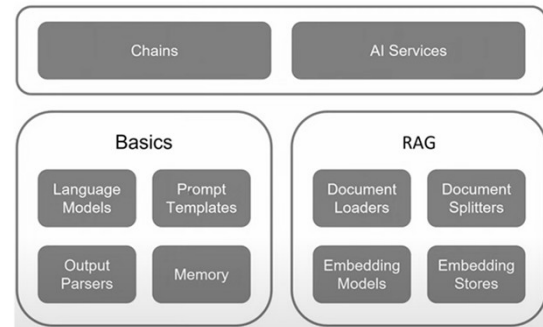
```
String cle = System.getenv("CLE_API");
```

R. Grin

LangChain4j

13

Composants de LangChain4j



R. Grin

LangChain4j

14

Types de modèles

- Interface `ChatModel` : représente un LM qui traite une requête et retourne une réponse
- Interface `StreamingChatModel` : comme `ChatModel` mais la réponse est retournée en streaming (token par token)
- Interface `EmbeddingModel` : pour convertir des textes en embeddings
- Interface `ImageModel` : générateur d'image à partir d'un texte
- Interface `ModerationModel` : surveillance des contenus échangés avec les LMs

R. Grin

LangChain4j

15

Interface ChatModel

- Package `dev.langchain4j.model.chat`
- 4 méthodes `chat` :
 - `ChatResponse chat(ChatRequest requête)`
 - `ChatResponse chat(List<ChatMessage> messages)`
 - `ChatResponse chat(ChatMessage... messages)`
 - `String chat(String userMessage)`
- `ChatResponse doChat(ChatRequest requête)` : méthode à implémenter pour un LM particulier

R. Grin

LangChain4j

16

Pourquoi plusieurs messages ?

- Les API sont sans état
- Si on veut que le modèle tienne compte des précédents messages de la conversation, il faut les passer à l'API à chaque requête

R. Grin

LangChain4j

17

ChatRequest, ChatResponse

- `ChatRequest` : Requête envoyée au LM ; peut contenir des messages (`ChatMessage`), le nom du modèle, les paramètres du LM à utiliser (`ChatRequestParameters`, température, informations sur les outils disponibles, ...), format de la réponse à utiliser (`ResponseFormat`), ...
- `ChatResponse` : réponse du LM, avec des informations complémentaires : métadonnées (`ChatResponseMetadata`), nom du modèle, tokens utilisés (`TokenUsage`), cause de l'arrêt de la réponse (`FinishReason`)

R. Grin

LangChain4j

18

Classe d'un modèle concret

- LangChain4j contient une API pour chacun des LMs qu'il supporte
- Chaque API implémente l'interface ChatModel
- Par exemple, avec GoogleAiGeminiChatModel pour Gemini et OpenAiChatModel pour OpenAI
- Une instance de la classe d'implémentation s'obtient et se configure le plus souvent avec le pattern « builder », mais il peut y avoir d'autres moyens (pattern « méthode fabrique static » par exemple)

R. Grin

LangChain4j

19

Pattern « builder »

- LangChain4j utilise très souvent ce pattern pour créer et configurer des instances
- Par exemple, pour les modèles concrets (pour Gemini ou OpenAI par exemple), les instances de ChatRequest, ChatRequestParameters, AiServices



R. Grin

LangChain4j

20

Méthodes chat de ChatModel

- ChatResponse chat(ChatRequest requête) : permet d'interroger le LM en utilisant les paramètres, les messages déjà dans requête et les informations (longueur max de la réponse, outils, ...) de la requête
- ChatResponse chat(List<ChatMessage> messages)
ChatResponse chat(ChatMessage... messages) : permet d'interroger le LM en utilisant les messages passés en paramètres
- String chat(String userMessage) : le message passé en paramètre et la réponse du LM sous la forme d'une simple String

R. Grin

LangChain4j

21

ChatRequest

- La question posée au modèle peut être une String mais aussi une ChatRequest pour prendre en compte des cas plus complexes
- Contient (voir Builder pour le nom du getter)
 - des messages (List<ChatMessage>)
 - des paramètres (ChatRequestParameters)
 - des spécifications d'outils (ToolSpecifications)
 - un format de réponse (ResponseFormat)

R. Grin

LangChain4j

22

Classe ChatRequest.Builder

- Retourné par ChatRequest.builder()
- Contient des méthodes qui retournent ChatRequest.Builder :
 - messages(ChatMessage... messages) et messages(List<ChatMessage> messages)
 - parameters(ChatRequestParameters parameters)
 - toolSpecifications(ToolSpecification... toolSpecifications) et toolSpecifications(List<ToolSpecification> toolSpecifications)
 - Nombreuses autres méthodes...

R. Grin

LangChain4j

23

Exemple simple pour Gemini

```
String cle = System.getenv("GEMINI_KEY");
// Création du modèle
ChatModel modele = GoogleAiGeminiChatModel
    .builder()
    .apiKey(cle)
    .modelName("gemini-2.5-flash")
    .build();

// Pose une question au modèle
String reponse = modele.chat("... ?");
System.out.println(reponse);
```

R. Grin

LangChain4j

24

Autre exemple avec Gemini

```
ChatModel modele =
    GoogleAiGeminiChatModel.builder()
        .apiKey(cle)
        .modelName("gemini-2.5-flash")
        .temperature(0.8)
        .timeout(Duration.ofSeconds(60))
        .responseFormat(ResponseFormat.JSON)
        .build();
ChatRequest requete = ChatRequest.builder()
    .temperature(0.5)
    .messages(SystemMessage.from("Réponds en bégayant"),
        UserMessage.from("Superficie de la France ?"))
    .build();
ChatResponse reponse = modele.chat(requete);
System.out.println(reponse.aiMessage().text());
System.out.println(reponse.tokenUsage().totalTokenCount());
```

R. Grin

LangChain4j

25

25

Exemple avec OpenAI

- Avec un builder pour créer le modèle :

```
ChatModel modele =
    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) :

```
ChatModel modele =
    OpenAiChatModel.withApiKey(cle);
```

R. Grin

LangChain4j

26

26

Interface ChatMessage

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

R. Grin

LangChain4j

27

27

Types de ChatMessage

- UserMessage : message de l'application ou de l'utilisateur qui peut contenir du texte, des images, de l'audio, de la vidéo ou du PDF
- AiMessage : message généré par le LM en réponse à un UserMessage ; peut contenir un texte (String, que l'on obtient par la méthode text()) ou une requête pour exécuter un outil (ToolExecutionRequest)
- SystemMessage : message pour le système ; décrit le rôle du LM, comment il doit se comporter, le style de réponse,... Souvent au début de la conversation
- ToolExecutionResultMessage : résultat d'une ToolExecutionRequest envoyé au LM

R. Grin

LangChain4j

28

28

Classe UserMessage

- Package dev.langchain4j.data.message
- Plusieurs façons de créer un UserMessage avec un texte (String), un ou plusieurs Content de différents types (image, audio, vidéo ou fichier PDF), et optionnellement un nom d'utilisateur :
 - constructeurs
 - méthode builder()
 - méthodes static from
 - méthodes static userMessage (comme from)
- Méthodes String singleText(), List<Content> contents() pour retrouver le contenu du message

R. Grin

LangChain4j

29

29

Interface Content

- Package dev.langchain4j.data.message
- Pour donner un des contenus d'un message utilisateur
- 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

R. Grin

LangChain4j

30

30

Multimodalité

- Tous les modèles n'acceptent pas tous les types de contenu pour poser une question
- Gemini et OpenAI acceptent les types image (ImageContent), vidéo (VideoContent), audio (AudioContent), PDF (PdfFileContent) ; toutes ces classes implémentent l'interface Content (classes et interfaces dans package dev.langchain4j.data.message)
- Par exemple, on peut demander à Gemini de décrire une image (voir exemple à suivre)

R. Grin

LangChain4j

31

31

Classe ImageContent

- Package dev.langchain4j.data.message
- Création avec un des constructeurs (choix parmi les paramètres de type Image, String ou URI pour l'URL, String pour l'image codée en base64, String pour le type Mime, ImageContent.DetailLevel pour niveau de détails de l'image) ou avec une des méthodes static from (avec les mêmes paramètres que les constructeurs)

R. Grin

LangChain4j

32

32

Exemple d'utilisation d'image

```
ChatResponse reponse =
    model.chat(UserMessage.from(
        ImageContent.from("https://.../uneimage.jpg"),
        TextContent.from("Décris-moi cette image.")
    ));
// Récupération du texte de la réponse
System.out.println(reponse.aiMessage().text());
```

R. Grin

LangChain4j

33

33

Classe Image

- Package dev.langchain4j.data.image
- Représente une image par son URL ou encodée en Base64
- Utilisé pour la génération d'image à partir d'un texte
- Pattern builder pour la création, avec les paramètres de type String pour le contenu de l'image codé en Base64, String pour le type Mime, String ou URI pour l'URL

R. Grin

LangChain4j

34

34

Classe SystemMessage

- Package dev.langchain4j.data.message
- Le plus souvent défini par le développeur, pas l'utilisateur
- Instructions sur le comportement, le style, le rôle du LM
- Les LMs 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

R. Grin

LangChain4j

35

35

AiMessage

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

R. Grin

LangChain4j

36

36



Exemples de messages

```
SystemMessage systemMessage =
    SystemMessage.from("Toutes tes réponses en majuscules");
UserMessage userMessage1 =
    UserMessage.from("Hello, mon nom est Julien");
AiMessage aiMessage1 =
    model.chat(systemMessage, userMessage1).aiMessage();
System.out.println(aiMessage1);
/* Affichage : { text = "BONJOUR JULIEN.", thinking = null,
toolExecutionRequests = [], attributes = {} } */
UserMessage userMessage2 = UserMessage.from("Mon nom ?");
AiMessage aiMessage2 =
    model.chat(userMessage1, aiMessage1, userMessage2)
        .aiMessage();
// { text = "Votre nom est Julien.", thinking = ... }
```

Pourquoi la dernière réponse n'est pas en majuscules ?

R. Grin

37

37

ChatRequestParameters

- Package dev.langchain4j.model.chat.request
- Regroupe les paramètres du LM : nom du modèle, hyperparamètres pour l'inférence (température, maxOutputTokens, utilisation outils, ...)
- Création avec le pattern builder
- Peuvent être spécifiés à la création du ChatModel (considérés alors comme les valeurs par défaut) ou bien attachés à un ChatRequest et utilisés à l'appel de la méthode chat
- `overrideWith(ChatRequestParameters)` permet de combiner avec d'autres valeurs de paramètres

R. Grin

LangChain4j

38

38

ToolChoice

- Énumération avec les valeurs
 - AUTO : le LM décide s'il utilise un ou plusieurs des outils
 - REQUIRED : le LM doit utiliser au moins un des outils

R. Grin

LangChain4j

39

39

DefaultChatRequestParameters

- Classe de dev.langchain4j.model.chat.request
- Implémentation par défaut de l'interface ChatRequestParameters

R. Grin

LangChain4j

40

40



Exemple

```
ChatRequestParameters parameters =
    ChatRequestParameters.builder()
        .modelName("gpt-4o-mini")
        .temperature(0.7)
        .build();
// Utilisation à la création du modèle
OpenAiChatModel model = OpenAiChatModel.builder()
    .apiKey(System.getenv("OPENAI_API_KEY"))
    .defaultRequestParameters(parameters)
    .build();
// Ou bien attaché à une requête particulière
ChatRequest chatRequest = ChatRequest.builder()
    .messages(UserMessage.from("Hello"))
    .parameters(parameters)
    .build();
```

R. Grin

LangChain4j

41

41

Interface ChatResponse

- Package dev.langchain4j.model.chat.response
- Réponse à un chat
- Méthodes :
 - `AiMessage aiMessage()`
 - `ChatResponseMetadata metadata()`
 - `TokenUsage tokenUsage()` : usage des tokens, nombres de tokens en entrée et en sortie
 - `FinishReason finishReason()` : une raison de fin de réponse
 - `String modelName()`

R. Grin

LangChain4j

42

42

Enumération FinishReason

- Package `dev.langchain4j.model.output`
- Indique pourquoi la génération du LM s'est arrêtée
- `STOP` : le LM 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 LM
- `OTHER` : autre raison

R. Grin

LangChain4j

43

Classe TokenUsage

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

R. Grin

LangChain4j

44

Exemple utilisation ChatResponse

```
UserMessage message = UserMessage.from("...");
ChatResponse reponse = modele.chat(message);

// Récupérer le texte de la réponse
String texteReponse = reponse.aiMessage().text();

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

R. Grin

LangChain4j

45

ChatResponseMetadata

- Interface du package `dev.langchain4j.model.chat.response`
- Représente les métadonnées habituelles supportées par les LMs ; chaque fournisseur de LM peut étendre cette interface
- Les métadonnées de base supportées (getters et méthodes du builder) : `finishReason`, `tokenUsage`, `modelName`

R. Grin

LangChain4j

46

Response<T>

- Classe du package `dev.langchain4j.model.output`
- Représente une réponse pour un modèle d'embeddings, un modèle de modération ou la génération d'image
- `T` : type de contenu généré par le modèle
 - `Embedding` pour un `EmbeddingModel`
 - `Moderation` pour un `ModerationModel`
 - `Image` pour un `ImageModel`

R. Grin

LangChain4j

47

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 LM

R. Grin

LangChain4j

48



PromptTemplate

- Package `dev.langchain4j.model.input`
- Pour obtenir des bonnes réponses à un LM, il faut poser les questions avec un bon format ; un template permet d'imposer un format prédéfini
- Une instance est créée avec la méthode `static from` ; on lui passe un modèle (de type `String`) qui peut contenir des variables, par exemple `{{nom}}`

R. Grin

LangChain4j

49

49

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

R. Grin

LangChain4j

50

50

Exemple

```
PromptTemplate template = PromptTemplate.from("""
Réponds en te basant sur les documents fournis :
{{context}}
Question : {{question}}""");

Prompt prompt = template.apply(Map.of(
    "context", "Les LLMs sont entraînés sur de grands
corpus de texte.",
    "question", "Comment fonctionne un LLM ?"
));
modele.chat(prompt.text())
```

R. Grin

LangChain4j

51

51

Variables prédéfinies

- `current_date` ; valeur `LocalDate.now()`
- `current_time` ; valeur `LocalTime.now()`
- `current_date_time` ; valeur `LocalDateTime.now()`

R. Grin

LangChain4j

52

52

Exemple avec variable prédéfinie

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

R. Grin

LangChain4j

53

53

Exemples avec variables

- `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"));`

R. Grin

LangChain4j

54

54

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: """);
```

R. Grin

LangChain4j

55

55

@StructuredPrompt

- Package `dev.langchain4j.model.input.structured`
- *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 template défini en plusieurs lignes, elles seront jointes avec ce délimiteur (\n par défaut)

R. Grin

LangChain4j

56

56

Exemple 1

```
@StructuredPrompt("Créer une recette de {{plat}} qui peut
être préparé en utilisant seulement {{ingredients}}")
public record PromptRecette(String plat,
    List<String> ingredients) { }

// Méthode main qui utilise PromptPourRecette :
PromptRecette prompt = new PromptRecette("salade",
    Arrays.asList("concombre", "tomate", "feta",
        "oignon", "olives"));

Prompt prompt =
    StructuredPromptProcessor.toPrompt(prompt);
String reponse = modele.chat(prompt.text());
System.out.println(reponse);
```

R. Grin

LangChain4j

57

57

Exemple 2 (plusieurs lignes)

```
@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:",
    "- ...",
    "- ..."
}) public record PromptRecette(..., ...) { }
```

Tout le reste est identique à l'exemple 1

R. Grin

LangChain4j

58

58

Interface ImageModel

- Package `dev.langchain4j.model.image`
- Pour générer une image à partir d'un texte
- Une méthode abstraite


```
Response<Image> generate(String prompt)
```
- Méthodes par défaut


```
Response<Image> generate(String prompt, int n):
    génère n images
Response<Image> edit(Image image, String
    prompt) : modifie l'image selon le prompt ; en 2ème
    paramètre on peut ajouter un masque (type Image) qui
    délimite la zone à modifier dans l'image
```

R. Grin

LangChain4j

59

59

Exemple avec Gemini (1/2)

```
ChatModel model = GoogleAiGeminiChatModel.builder()
    .modelName("gemini-2.5-flash-image-preview")
    .apiKey(llmKey).build();
String descriptionImage = "...";
ChatResponse reponse =
    model.chat(UserMessage.from(descriptionImage));
AiMessage aiMessage = reponse.aiMessage();
List<Image> images =
    GeneratedImageHelper.getGeneratedImages(aiMessage);
if (GeneratedImageHelper.hasGeneratedImages(aiMessage)) {
    System.out.println(images.size() + " images générées");
    System.out.println("Texte de la réponse : "
        + aiMessage.text());
}
```

Réservé version payante API Gemini

R. Grin

LangChain4j

60

60

Exemple avec Gemini (2/2)

```
int n = 1;
for (Image image : images) {
    String base64Data = image.base64Data();
    String mimeType = image.mimeType();
    // On peut sauvegarder afficher, modifier image.
    // Pour la sauvegarder :
    byte[] imageBytes =
        Base64.getDecoder().decode(base64Data);
    Files.write(Paths.get("image" + n++ + ".png"),
        imageBytes);
}
} else {
    System.out.println(aiMessage.text());
}
```

R. Grin

LangChain4j

61

61

Exemple avec OpenAI

```
import dev.langchain4j.data.image.Image;
import dev.langchain4j.model.image.ImageModel;
import dev.langchain4j.model.openai.OpenAiImageModel;
import dev.langchain4j.model.output.Response;
import static
dev.langchain4j.model.openai.OpenAiImageModelName.DALL_E_3;
...
ImageModel model = OpenAiImageModel.builder()
    .apiKey(ApiKeys.OPENAI_API_KEY)
    .modelName(DALL_E_3)
    .build();
Response<Image> response = model.generate("...");
System.out.println(response.content().url());
}
```

R. Grin

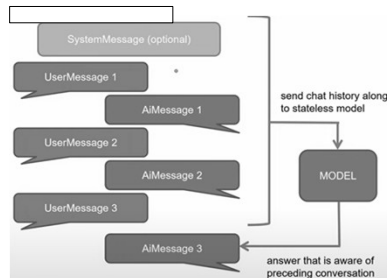
LangChain4j

62

62

Etat de la conversation

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



R. Grin

LangChain4j

63

63

Etat et historique conversation

- Dans l'état de la conversation avec le modèle, on peut vouloir conserver des informations différentes des messages exacts échangés
- On peut, par exemple, ne pas conserver des messages peu importants, ou ne conserver qu'un résumé de certains messages, ou de ce qu'il s'est passé pendant la conversation
- L'enregistrement se fait à part, explicitement, dans une instance de type ChatMemory

R. Grin

LangChain4j

64

64

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

R. Grin

LangChain4j

65

65

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 ; peut servir à distinguer 2 conversations qui se déroulent en parallèles

R. Grin

LangChain4j

66

66

Implémentations ChatMemory

- Dans package `dev.langchain4j.memory.chat` :
 - `MessageWindowChatMemory` : ne retient que les N derniers messages
 - `TokenWindowChatMemory` : ne retient que les N derniers tokens ; nécessite un `Tokenizer` pour évaluer le nombre de 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 `SingleSlotChatMemoryStore` est utilisé (conserve en mémoire centrale)

R. Grin

LangChain4j

67

67

Création d'une mémoire

- Les 2 classes fournies utilisent le pattern « builder »
- Le builder permet
 - de donner un id (de type `Object`) pour distinguer plusieurs mémoires
 - d'indiquer le type de `ChatMemoryStore` à utiliser pour conserver les messages
 - de fixer le nombre maximum de messages ou de tokens à conserver
- Le plus souvent on crée une instance en fixant le nombre maximum de messages ou de tokens avec les méthodes `static withMaxMessages` ou `withMaxTokens`, suivant la classe

R. Grin

LangChain4j

68

68

Exemple

```
ChatMemory memoire =
    MessageWindowChatMemory.withMaxMessages(10);
String question = "Films dirigés par Spielberg ?";
memoire.add(UserMessage.from(question));
ChatResponse reponse = modele.chat(memoire.messages());
// AiMessage extrait de la réponse et ajouté à la mémoire
memoire.add(reponse.aiMessage());
// Nouvelle question
question = "Quel âge a-t-il ?";
memoire.add(UserMessage.from(question));
reponse = modele.chat(memoire.messages());
System.out.println(reponse.aiMessage().text());
```

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

R. Grin

LangChain4j

69

69

@MemoryId

- Annotation du package `dev.langchain4j.service`, qui peut se mettre sur un des paramètres d'une méthode d'une interface utilisée par AI services
- La valeur du paramètre annoté sera utilisée pour identifier la mémoire qui est associée à une conversation/un utilisateur
- Le paramètre peut être de n'importe quel type
- Le `ChatMemoryProvider` doit être configuré pour les AI service (voir code précédent)

R. Grin

LangChain4j

70

70

Interface ChatMemoryStore

- Package `dev.langchain4j.store.memory.chat`
- Implémentée par plusieurs classes dont `InMemoryChatMemoryStore`, `CassandraChatMemoryStore`
- Possible de récupérer des messages, de les supprimer ou de les modifier (utilise un id associé à chaque mémoire)
- Il est simple de créer une mémoire personnalisée en implémentant l'interface `ChatMemoryStore` ; par exemple pour enregistrer la mémoire dans une base de données relationnelle et ainsi garder l'historique en mémoire au-delà d'une seule session

R. Grin

LangChain4j

71

71

Utilisation mémoire persistante


```
ChatMemory chatMemory = MessageWindowChatMemory.builder()
    .id("12345")
    .maxMessages(10)
    .chatMemoryStore(new PersistentChatMemoryStore())
    .build();
```

R. Grin

LangChain4j

72

72



Trace et logging pour la sécurité

- Confier du travail à un logiciel qui peut halluciner n'est pas sans risques !
- Il est donc indispensable de garder une trace des échanges avec les parties des programmes qui utilisent des LMs, pour
 - Lire et évaluer leurs réponses
 - Comprendre pourquoi une erreur s'est produite
 - Améliorer leur comportement

R. Grin LangChain4j 73

73

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)

R. Grin LangChain4j 74

74

Exemple de logging

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

R. Grin LangChain4j 75

75

Configuration logging

- Pour que le logging fonctionne, il faut le configurer (sinon un avertissement s'affiche : « SLF4J(W): No SLF4J providers were found. SLF4J(W): Defaulting to no-operation (NOP) logger implementation »)
- 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 ➡

R. Grin LangChain4j 76

76

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>
```

R. Grin LangChain4j 77

77

Configuration par code pour JUL

```
import java.util.logging.Logger;
...
private static void configureLogger() {
    // Configure le logger sous-jacent (java.util.logging)
    // pour les types de LangChain4j
    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);
}
```

R. Grin LangChain4j 78

78

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

R. Grin

LangChain4j

79

79

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

R. Grin

LangChain4j

80

80

Types d'embeddings

- 2 grands types d'embeddings que l'on peut utiliser avec LangChain4j :
 - Modèles qui s'exécutent à part (distants ou locaux) et que l'on utilise avec une API ; par exemple GoogleAiEmbeddingModel, OpenAiEmbeddingModel, CohereEmbeddingModel, HuggingFaceEmbeddingModel
 - Modèles intégrés (« in-process ») qui s'exécutent dans le même processus que le code qui les utilise ; par exemple AllMiniLmL6V2EmbeddingModel ou AllMiniLmL6V2QuantizedEmbeddingModel

R. Grin

LangChain4j

81

81

Exemple modèle d'embeddings

```

EmbeddingModel modele = GoogleAiEmbeddingModel.builder()
    .apiKey(llmKey)
    .modelName("gemini-embedding-001")
    .taskType(GoogleAiEmbeddingModel.TaskType.SEMANTIC_SIMILARITY)
    .outputDimensionality(300)
    .timeout(Duration.ofSeconds(2))
    .build();

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 similarite = CosineSimilarity.between(emb1, emb2);
System.out.println("Similarité cosinus : " + similarite);

```

R. Grin

LangChain4j

82

82

Exemple modèle intégré

```

// Même code que l'exemple précédent, sauf pour la
// création du modèle d'embedding :
EmbeddingModel modele =
    new AllMiniLmL6V2EmbeddingModel();

```

R. Grin

LangChain4j

83

83

AiServices

R. Grin

LangChain4j

84

84

Chaînes et services IA

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

R. Grin

LangChain4j

85

85

Service IA

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

R. Grin

LangChain4j

86

86

Exemple d'interface

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

On peut choisir le nom des méthodes de l'interface

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

R. Grin

LangChain4j

87

87

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

R. Grin

LangChain4j

88

88

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 (peut être gérée automatiquement)
 - RAG (RetrievalAugmentor et ContentRetriever)
 - Outils (Tools)
 - Streaming
 - Auto-modération
 - ...



R. Grin

LangChain4j

89

89

Méthodes de AiServices<T> (1/4)

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

R. Grin

LangChain4j

90

90

Méthodes de AIServices<T> (2/4)

- `public AIServices<T> chatModel(ChatModel model)` : indique le modèle qui sera utilisé par le service IA
- `public AIServices<T> streamingChatModel(StreamingChatModel 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)

R. Grin

LangChain4j

91

91

Memory pour chaque utilisateur

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

Assistant assistant = AIServices.builder(Assistant.class)
    .chatModel(OpenAiChatModel.withApiKey(...))
    .chatMemoryProvider(memoryId ->
        MessageWindowChatMemory.withMaxMessages(10))
    .build();

System.out.println(
    assistant.chat(1, "Hello, my name is Jean"));
// Hi Jean! 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 Jean.

System.out.println(assistant.chat(2, "What is my name?"));
// Your name is Francine.
```

R. Grin

LangChain4j

92

92

Méthodes de AIServices<T> (3/4)

- `public AIServices<T> tools(List<Object> objectsWithTools)` : configure les outils que le LM pourra utiliser ; 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`)

R. Grin

LangChain4j

93

93

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

R. Grin

LangChain4j

94

94



Exemple simple

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

Assistant assistant = AIServices.builder(Assistant.class)
    .chatModel(modele)
    .build();

String reponse = assistant.chat("Hello, world!");
System.out.println(reponse);
```

R. Grin

LangChain4j

95

95

Exemple simple avec mémoire

```
ChatModel modele = GoogleAiGeminiChatModel.builder()
    .apiKey(cle)
    .modelName("gemini-2.5-flash")
    .build();

ChatMemory memoire =
    MessageWindowChatMemory.withMaxMessages(10);

Assistant assistant = AIServices.builder(Assistant.class)
    .chatModel(modele)
    .chatMemory(memoire)
    .build();

String rep1 = assistant.chat("Hello! je m'appelle Julie");
System.out.println(rep1); // Hello Julie !

String rep2 = assistant.chat("Quel est mon nom?");
System.out.println(rep2); // Votre nom est Julie.
```

R. Grin

LangChain4j

96

96

Comment ça marche ?

- AiServices crée un objet proxy qui implémente l'interface (en utilisant la réflexivité)
- Le proxy s'occupe, entre autres,
 - d'envoyer les requêtes au LM dans un format compatible avec le LM (souvent JSON)
 - de recevoir les réponses du LM (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

R. Grin

LangChain4j

97

97

Méthodes de l'interface

- Elles définissent les interactions avec le service IA
- Elles peuvent être annotées par @UserMessage ou @SystemMessage pour définir le texte d'un UserMessage ou d'un SystemMessage qui sera envoyé au LM dans chaque requête ; @UserMessage peut aussi être mis sur un paramètre
- Elles ont des contraintes sur les paramètres et le type retour car elles correspondent à une requête qui est envoyée au LM

R. Grin

LangChain4j

98

98

Paramètres des méthodes

- Les paramètres des méthodes sont de type String (ou Content pour les requêtes multimodales)
- Ils peuvent correspondre au texte envoyé au LM (UserMessage)
- Le texte du UserMessage (ou SystemMessage) peut être un template ; les valeurs des variables du template sont alors définies par les paramètres de la méthode de type String annotés par @V

R. Grin

LangChain4j

99

99

Cas simple de méthode

- `String chat(String userMessage)`
- Un seul paramètre pour un UserMessage



R. Grin

LangChain4j

100

100

Exemples de méthodes (1/3)

- `String chat(@UserMessage String userMessage)` puisqu'il n'y a qu'un seul paramètre, identique à `String chat(String userMessage)`
- `@SystemMessage("Donner le nom de la capitale du pays indiqué, et rien d'autre")`
`String chat(String pays);`
`@SystemMessage` peut aussi s'appliquer à un paramètre s'il peut changer dynamiquement
- `@UserMessage("Valeur de {{info}} sur {{pays}} ?")`
`String chat(@V("info") String info, @V("pays") String pays);`
`@UserMessage` contient un template. Les paramètres du template sont les paramètres de la méthode, annotés par @V

R. Grin

LangChain4j

101

101

Exemples de méthodes (2/3)

- `@UserMessage("Valeur de {{info}} sur {{pays}} ?")`
`String chat(@V("info") String info, @V("pays") String pays);`
`@UserMessage` contient un template. Les paramètres du template sont les paramètres de la méthode, annotés par @V
On pourra l'utiliser comme ceci :
`assistant.chat(population, France);`
- Multimodalité :
`String chat(@UserMessage String userMessage, @UserMessage ImageContent image, @UserMessage AudioContent audio);`

R. Grin

LangChain4j

102

102

Exemples de méthodes (3/3)

- `@SystemMessage("You are a helpful assistant that always replies in French.")`
`String chat(@MemoryId String sessionId,`
`@UserMessage String userMessage`
`);`
- `@SystemMessage("Donner le nom de la capitale du pays indiqué, et rien d'autre")`
`@UserMessage("{pays}") // template...`
`String chat(@V("pays") String pays);`
- `@SystemMessage("Pour le pays donné, {{question}}")`
`@UserMessage("{pays}")`
`String chat(@V("question") String question,`
`@V("pays") String pays);`

R. Grin

LangChain4j

103

103

Exemple complet

```
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)
    .chatModel(modele).build();
```

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

R. Grin

LangChain4j

104

104

Types retour méthodes interface

- `String` ou `ChatResponse` ; `ChatResponse` permet d'avoir en plus `TokenUsage`, métadonnées ou `FinishReason`
- Nombreux autres types possibles :
 - `List<String>` ou `Set<String>`
 - `Map<K,V>`
 - Une énumération ou un `boolean` (par exemple si on veut utiliser le LM 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

R. Grin

LangChain4j

105

105

Type `Result<T>`

- `Result<T>` (package `dev.langchain4j.service`) pour envelopper un autre type supporté par les services IA ; permet d'avoir des informations supplémentaires sur le traitement de la requête : `FinishReason`, `TokenUsage`, les sources (`List<Content>`) si RAG, les outils utilisés (`List<ToolExecution>`) ; `content()` donne la réponse du LM (de type `T`) ; `finalResponse()` donne la `ChatResponse` finale et `intermediateResponses()` donne toutes les réponses intermédiaires du LM
- `T` est le type supporté par les services IA (`AIServices`) en type retour : `String`, `Enum`, ...

R. Grin

LangChain4j

106

106

Exemple avec `Result<T>`

```
public interface AgentMeteo {
    Result<String> chat(String requete);
}
```

```
AgentMeteo agentMeteo =
    AIServices.builder(AgentMeteo.class)
        .chatModel(model)
        .tools(new MeteoTool()) // Ajout outil
        .build();

Result<String> resultat =
    agentMeteo.chat("Quel temps fait-il à Nice ?");
System.out.println(resultat.content()); // Réponse LM
TokenUsage tokenUsage = resultat.tokenUsage();
List<ToolExecution> executions = resultat.toolExecutions();
for (ToolExecution toolExecution : executions) {
    System.out.println(toolExecution.toString());
}
```

R. Grin

LangChain4j

107

107

Extraction de données

R. Grin

LangChain4j

108

108

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 standards 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

R. Grin

LangChain4j

109

109

ExtracteurDateTemps (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);
}
```

R. Grin

LangChain4j

110

110

ExtracteurDateTemps (2/2)

```
ExtracteurDateTemps extracteur =
    AIServices.builder(ExtracteurDateTemps.class)
        .chatModel(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);
```

R. Grin

LangChain4j

111

111

Pour les classes Java

- L'extraction fonctionne aussi sous la forme de classes ou de records Java, créés dans l'application
- Voici, par exemple, un 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.
- Voyons comment récupérer dans un record Java, le nom, les dates et lieux de naissance du personnage dont on parle

R. Grin

LangChain4j

112

112

Etapes

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

R. Grin

LangChain4j

113

113

Exemple, étape 1

```
ChatModel modele =
    GoogleAiGeminiChatModel
        .builder()
        .apiKey(llmKey)
        .modelName("gemini-2.5-flash")
        .responseFormat(ResponseFormat.JSON)
        .build();
```

R. Grin

LangChain4j

114

114

Exemple, étape 2

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

R. Grin

LangChain4j

115

115

Exemple, étape 3

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

Que faut-il modifier si on veut aussi avoir l'usage des tokens ?

R. Grin

LangChain4j

116

116

Exemple, étape 4

```
ExtracteurInfosPersonne extracteur =
    AIServices.builder(ExtracteurInfosPersonne.class)
        .chatModel(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());
...
```

R. Grin

LangChain4j

117

117

Description pour extraction

- Si le type retour est un type structuré Java et si les contenus des attributs ne sont pas clairs d'après leur nom, on peut ajouter sur un champ ou sur le type, une annotation `@Description` qui précise les choses et aidera le LM à faire son travail
- Par exemple


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

R. Grin

LangChain4j

118

118

Réponse structurée avec métadonnées

- Pour avoir en plus les informations sur les tokens, les sources (si RAG), les exécutions d'outils,...), il suffit d'envelopper le type retour Java avec `Result` :


```
Result<Personne> extraireInfosPersonne(String
            texte);
```

R. Grin

LangChain4j

119

119

Outils

R. Grin

LangChain4j

120

120

Appel de fonction des LMs

- Les LMs ont des lacunes ; par exemple ils ne sont pas bons en mathématiques et ils ne peuvent pas consulter Internet en temps réel
- Pour combler ces lacunes, de nombreux LMs, dont OpenAI et Gemini, ont ajouté la possibilité de faire appel à des outils qui sont exécutés dans le processus de génération
- Ces outils étendent les capacités des LMs en leur donnant accès à des fonctionnalités spécifiques ou des ressources externes

R. Grin

LangChain4j

121

121

Outil

- Code écrit dans un langage informatique quelconque qui peut aider un LM à générer sa réponse
- Situation : Un agent (par exemple ChatGPT ou le code Java écrit dans les TPs) utilise un LM (par exemple GPT-5)
- Pour qu'un LM puisse utiliser un outil, l'agent doit d'abord lui présenter cet outil (lui décrire ce qu'il fait et comment l'appeler) afin que le modèle sache quand et comment s'en servir
- Si le LM juge qu'un outil lui permettra de répondre à une question, il demande à l'agent de l'exécuter et de lui fournir le résultat de l'exécution

R. Grin

LangChain4j

122

122

Exemples d'outils

- Recherche Web pour obtenir des informations actualisées (par consultation de moteurs de recherche)
- Accès à des API spécialisées ; par exemple API pour la météo ou pour la conversion de devises
- Accès à des bases de données
- Calcul numérique ou formel
- Exécution de code ; par exemple interpréteur Python
- Synthèse et reconnaissance vocale
- Envoyer et gérer des emails

R. Grin

LangChain4j

123

123

Processus utilisation outils

1. L'agent envoie une requête en indiquant quels outils sont disponibles pour aider à répondre
2. Le LM détecte quels outils il souhaite utiliser pour répondre à la requête et, dans une réponse intermédiaire, il donne les informations nécessaires à l'exécution de ces outils : endpoint, paramètres, ...
3. Les outils sont exécutés par le client du LM
4. Une requête envoie au LM le résultat de l'exécution
5. Le LM utilise le résultat pour générer la réponse à la requête initiale

R. Grin

LangChain4j

124

124

Exemple

1. On envoie une question au LM en lui indiquant qu'il peut utiliser un outil pour faire des calculs complexes et un outil pour envoyer des emails
2. Pour répondre à un prompt, le LM « réfléchit » et s'aperçoit qu'il a besoin de l'outil pour faire un calcul complexe ; dans une 1^{ère} réponse à la question, le LM demande de faire le calcul complexe pour lui, en passant les paramètres du calcul
3. Le calcul est exécuté en local
4. Le résultat est passé au LM par une requête spéciale
5. Le LM utilise ce résultat pour générer sa réponse

R. Grin

LangChain4j

125

125

Etapes (1/2)

- Ecrire les outils dans un langage informatique
 - 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 LM pour savoir si la fonction lui sera utile pour répondre à la question
1. Envoyer une 1^{ère} requête au LM, avec une question et des descriptions des outils (appelés aussi fonctions)
 2. Il répond en indiquant quels outils il utilisera pour répondre à la question ; la réponse contiendra un champ « functionCall » (demande d'exécution)

R. Grin

LangChain4j

126

126

Exemple réponse LLM

```
{
  "candidates": [
    {
      "content": {
        "parts": [
          {
            "functionCall": {
              "name": "dateLivraison",
              "args": { "arg0": "12345" }
            }
          }
        ],
        "role": "model"
      }
    }
  ]
}
```

Le LM a besoin de l'outil « dateLivraison » pour lui indiquer quand la commande d'id 12345 sera livrée

R. Grin

LangChain4j

127

127

Etapes (2/2)

3. En local, de la réponse du LM à la 1^{ère} requête, l'agent extrait les noms des outils avec les valeurs de leurs paramètres, et il exécute les fonctions avec leurs paramètres
4. L'agent envoie une 2^{ème} requête au LM, avec les résultats de l'exécution des fonctions dans un message dont le rôle est « tool »
5. Ce résultat est utilisé par le LM pour donner sa réponse à la question

R. Grin

LangChain4j

128

128

Exemple requête avec résultat

```
{
  "parts": [ {
    "functionResponse": {
      "name": "dateLivraison",
      "response": {
        "response": "<date retournée par outil>"
      }
    }
  } ],
  "role": "user"
}
```

L'agent IA fournit le résultat de l'exécution de l'outil

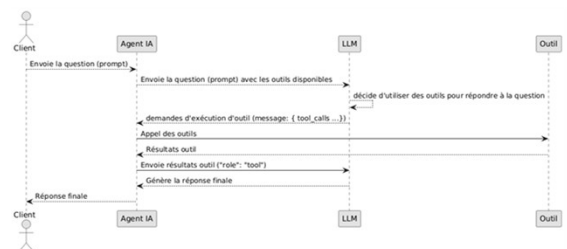
R. Grin

LangChain4j

129

129

Diagramme de séquence



R. Grin

LangChain4j

130

130

Apport de LangChain4j

- Le processus qui vient d'être décrit est complexe :
 1. Il faut envoyer 2 requêtes
 2. Extraire des informations de la 1^{ère} requête
 3. Utiliser ces informations pour appeler la fonction
 4. Donner le résultat de l'exécution dans la 2^{ème} 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

R. Grin

LangChain4j

131

131

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 $\sqrt{14}$)

R. Grin

LangChain4j

132

132

Exemple - le code

```
Assistant assistant = AIServices.builder(Assistant.class)
    .chatModel(modele)
    .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);
}
```

R. Grin

LangChain4j

133

133

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);
    }
}
```

R. Grin

LangChain4j

134

134

@P

- Package dev.langchain4j.agent.tool
- Donne une description en langage naturel d'un paramètre d'un outil, pour aider le LM à choisir les outils qu'il va utiliser et à les utiliser correctement
- En plus de l'attribut value, l'annotation @P peut avoir un attribut required pour indiquer si le paramètre est requis (requis par défaut) ; s'il est requis, le LM devra fournir ce paramètre pour utiliser l'outil

R. Grin

LangChain4j

135

135

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) {
        ...
    }
}
```

R. Grin

LangChain4j

136

136

Exemple avec devises (1/2)

```
@Tools("Convertit un montant monétaire d'une devise devise1 vers une autre devise devise2 ")
public double convert(double montant, @P("devise1")
String devise1, @P("devise2") String devise2) {
    return montant * taux.get(devise1) / taux.get(devise2);
}

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

R. Grin

LangChain4j

137

137

Exemple avec devises (2/2)

```
String question = ""
    Sur le site Vente.com de plusieurs pays, un ordinateur coûte 718,25 GBP, 83900 INR, 749,99 USD, et 177980 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);
```

R. Grin

LangChain4j

138

138

@Tool

- Package `dev.langchain4j.agent.tool`
- 2 paramètres optionnels :
 - `name` est le nom de l'outil ; par défaut le nom de la méthode de l'outil)
 - `value` est la description de l'outil ; par défaut déduit du nom de la méthode

R. Grin

LangChain4j

139

139

@Description

- Package `dev.langchain4j.model.output.structured`
- Peut annoter une classe, ou un champ d'une telle classe, en particulier d'une classe d'outils
- Peut aider le LM à choisir les outils qu'il va utiliser et à les utiliser correctement
- On a vu que cette annotation peut aussi servir pour décrire des champs de structures de données dans l'extraction des données

R. Grin

LangChain4j

140

140

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) {
    ...
}
```

R. Grin

LangChain4j

141

141

Types paramètres des outils

- Types primitifs
- `String`, enveloppes des types primitifs
- Enumérations
- Classe quelconque (doit permettre la conversion en JSON)
- `List<T>`, `Set<T>` (avec T d'un des types ci-dessus)
- `Map<K, V>` (on doit spécifier les types K et V dans la description du paramètre avec @P)
- L'idée est que le type doit pouvoir être converti en JSON

R. Grin

LangChain4j

142

142

Type retour des outils

- Si le type retour est `String`, la valeur retournée est envoyée au LM
- Si le type retour est `void`, la chaîne « Success » est envoyée au LM
- Sinon, la valeur retournée est sérialisée en JSON et envoyée au LM

R. Grin

LangChain4j

143

143

Gestion des erreurs dans outils

- Si une méthode annotée avec `@Tool` lève une exception, le message de l'exception (e.g. `getMessage()`) sera envoyé au LM à la suite de l'exécution de l'outil
- Le LM pourra ainsi corriger son erreur et réessayer, s'il le juge nécessaire

R. Grin

LangChain4j

144

144

@ToolMemoryId

- Si un paramètre d'une méthode d'un outil est annoté avec cette annotation, le paramètre aura la valeur du memory id (la mémoire de la conversation ; voir [AI services](#))
- Un état pourra ainsi être maintenu entre les appels de l'outil

R. Grin

LangChain4j

145

145

Définir outil par programmation

- Pour les cas particuliers où les outils sont chargés pendant l'exécution depuis des sources externes, on peut les définir en utilisant ToolSpecification
- Il faut alors fournir une implémentation de ToolExecutor

R. Grin

LangChain4j

146

146

Exemple - ToolSpecification

```
ToolSpecification toolSpecification =
    ToolSpecification.builder()
        .name("get_booking_details")
        .description("Returns booking details")
        .parameters(JsonObjectSchema.builder()
            .properties(Map.of(
                "bookingNumber",
                JsonStringSchema.builder()
                    .description("Booking number in B-12345 format")
                    .build()
            ))
            .build()
        ).build();
```

R. Grin

LangChain4j

147

147

Exemple - ToolExecutor

```
ToolExecutor toolExecutor =
    (toolExecutionRequest, memoryId) -> {
        Map<String, Object> arguments =
            fromJson(toolExecutionRequest.arguments());
        String bookingNumber =
            arguments.get("bookingNumber").toString();
        Booking booking = getBooking(bookingNumber);
        return booking.toString();
    };
```

R. Grin

LangChain4j

148

148

Exemple - création service IA

```
Assistant assistant = AiServices.builder(Assistant.class)
    .chatModel(chatModel)
    .tools(Map.of(toolSpecification, toolExecutor))
    .build();
```

R. Grin

LangChain4j

149

149

Définir outils dynamiquement

- Des outils peuvent être ajoutés dynamiquement, selon le contenu de chaque requête
- Il faut pour cela utiliser un ToolProvider

R. Grin

LangChain4j

150

150

Appel d'outil inexistant

- Pendant le traitement d'une requête le LM peut halluciner et demander l'utilisation d'un outil inexistant
- Par défaut une exception sera alors lancée
- On peut changer ce comportement par défaut en définissant une stratégie avec la méthode `hallucinatedToolNameStrategy` du builder de `AiServices`, au moment de la création du service IA

R. Grin

LangChain4j

151

151



Modération

R. Grin

LangChain4j

152

152

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

R. Grin

LangChain4j

153

153

@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

R. Grin

LangChain4j

154

154

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

R. Grin

LangChain4j

155

155

Interface `ModerationModel`

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

R. Grin

LangChain4j

156

156

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

R. Grin

LangChain4j

157

Exemple (1/3)

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

R. Grin

LangChain4j

158

Exemple (2/3)

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

ChatModel chatModel = ...;

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

R. Grin

LangChain4j

159

Exemple (3/3)

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

R. Grin

LangChain4j

160

Streaming

R. Grin

LangChain4j

161

Introduction

- Les modèles génèrent leurs réponses token par token (ou groupe de tokens par groupes de tokens)
- 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 StreamingChatModel, à la place de ChatModel
- Le développeur doit implémenter un handler pour indiquer ce qui se passera quand l'application recevra les tokens



R. Grin

LangChain4j

162

StreamingChatModel

- Interface du package `dev.langchain4j.model.chat`
- 3 méthodes `chat` qui retournent `void` et ont un `StreamingChatResponseHandler` en dernier paramètre
- Le 1^{er} paramètre représente le ou les messages de la conversation : `ChatRequest`, `String` (pour `userMessage`) ou `List<ChatMessage>`
- Méthode `void doChat(ChatRequest, StreamingChatResponseHandler)`
- Méthode `List<ChatModelListener> listeners()`
- Méthode `Set<Capability> supportedCapabilities()`
- Implémentations : `OpenAiStreamingChatModel`, `GoogleAiGeminiStreamingChatModel`, ...

R. Grin

LangChain4j

163

163

Streaming avec AI services

- Les AI services permettent de faire simplement du streaming
- Il faut utiliser `TokenStream` comme type retour des méthodes de l'interface de l'assistant IA

R. Grin

LangChain4j

164

164

Interface TokenStream

- Package `dev.langchain4j.service`
- Représente un stream de tokens
- La méthode `start()` démarre l'envoi de la requête au LM et le traitement des tokens émis par le LM en réponse
- On peut recevoir des notifications
 - quand une nouvelle réponse partielle est disponible ; la méthode `onPartialResponse` prend en paramètre comment sera consommée cette réponse (de type `String`)
 - quand le LM a terminé sa réponse ; `onCompleteResponse` prend en paramètre comment sera consommée la `ChatResponse`
 - quand une erreur est survenue pendant le streaming (`onError`)

R. Grin

LangChain4j

165

165

Exemple

```
TokenStream tokenStream = assistant.chat(question);
tokenStream
    .onPartialResponse(tokens ->
        { /* Traitement du groupe de tokens (type String)
          qui vient d'arriver */ })
    .onCompleteResponse(chatResponse ->
        { /* Traitement de la ChatResponse quand la réponse
          est complète */ })
    .onError(erreur ->
        { /* Traitement de la Throwable */ })
    .start(); // Démarre le traitement de la question
              // et réception/traitement du stream de tokens
```

R. Grin

LangChain4j

166

166

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)
    .streamingChatModel(modele)
    .chatMemory(chatMemory)
    .build();
```

R. Grin

LangChain4j

167

167

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

```
TokenStream tokenStream = assistant.chat("...");
CompletableFuture<ChatResponse> future =
    new CompletableFuture();
tokenStream
    .onPartialResponse(System.out::println)
    .onCompleteResponse(future::complete)
    .onError(future::completeExceptionally)
    .start();
ChatResponse chatResponse = future.get(30, SECONDS);
System.out.println("\n" + chatResponse);
```

R. Grin

LangChain4j

168

168

Méthodes de TokenStream (1/3)

- `TokenStream onPartialResponse(Consumer<String> partialResponseHandler)` appelée à chaque nouvelle réponse partielle disponible
- `TokenStream onCompleteResponse(Consumer<ChatResponse> completeResponseHandler)` appelée quand le LM a fini de répondre (fin du streaming)
- `TokenStream onError(Consumer<Throwable> errorHandler)` appelée quand erreur dans streaming
- `void start()` finit la construction du stream et démarre le streaming

R. Grin

LangChain4j

169

169

Méthodes de TokenStream (2/3)

- `TokenStream onToolExecuted(Consumer<ToolExecution> toolExecuteHandler)` appelée après qu'un outil a fini son exécution (et avant l'exécution d'un autre outil)
- `TokenStream ignoreErrors()` toutes les erreurs seront ignorées (mais loggées avec le niveau WARN)
- `default TokenStream onRetrieved(Consumer<List<Content>> contentHandler)` appelée quand un Content est retrouvé par un `RetrievalAugmentor`
- `default TokenStream beforeToolExecution(Consumer<BeforeToolExecution handler>)`

R. Grin

LangChain4j

170

170

Méthodes de TokenStream (3/3)

- `default TokenStream onRetrieved(Consumer<List<Content>> contentHandler)` appelée quand un Content est retrouvé par un `RetrievalAugmentor`
- `default TokenStream beforeToolExecution(Consumer<BeforeToolExecution handler>)` appelée juste avant l'exécution d'un outil
- `default TokenStream onPartialThinking(Consumer<PartialThinking handler>)` appelée quand une réponse partielle d'un texte « thinking » est reçu

R. Grin

LangChain4j

171

171

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

R. Grin

LangChain4j

172

172

APIs et outils autour de IA

R. Grin

LangChain4j

173

173

LlamaIndex

- Comme LangChain, facilite l'intégration des LMs 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 LMs 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.

R. Grin

Fine-tuning et RAG

174

174

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 LMs
- Offre un ensemble d'outils pour optimiser chaque étape du projet
- Pour débogage, collaboration avec une équipe de développement, tests et surveillance des applications

R. Grin

LangChain4j

175

175

LangSmith (2/2)

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

R. Grin

LangChain4j

176

176

- Whisper de OpenAI pour « speech 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??*???

R. Grin

LangChain4j

177

177

- 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>

R. Grin

LangChain4j

178

178

Autres liens intéressants

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

R. Grin

LangChain4j

179

179

- 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=vRvlqEQGLzQ&list=PLZOgUaAUCiT7ooFAUWld7oeWatv6b3oCD>

R. Grin

LangChain4j

180

180

ElevenLabs

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

R. Grin

LangChain4j

s81

181

Outils pour écrire du code

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

R. Grin

LangChain4j

s82

182

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 chatter 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

R. Grin

LangChain4j

s83

183

Erreurs avec outils IA pour coder

- Référence de fichiers qui n'existent pas
- Code qui utilise d'anciennes versions des bibliothèques
- 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

R. Grin

LangChain4j

s84

184

Références

R. Grin

LangChain4j

s85

185

IA avec Java (1/2)

- « AI for Java developers » par Microsoft : <https://www.youtube.com/watch?v=V45tKEYYAFs&list=PLPeZXICR7ew8sdUWqfzitrRG5BUE7GFsy>
- 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> <https://deeplearning4j.konduit.ai/>
- LangChain4j, <https://github.com/langchain4j/langchain4j>

R. Grin

LangChain4j

s86

186

IA avec Java (2/2)

- Playlist YouTube sur IA en Java : <https://www.youtube.com/playlist?list=PLZOgUaAUCiT7ooFAUWId7oeWatv6b3oCD>

R. Grin

LangChain4j

187

187

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>
- Projet de SDK d'OpenAI : <https://github.com/openai/openai-java>

R. Grin

LangChain4j

188

188

API de Gemini

- Documentation sur l'API : <https://ai.google.dev/gemini-api/docs>
- Modèles de l'API : <https://ai.google.dev/gemini-api/docs/models>

R. Grin

LangChain4j

189

189

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>

R. Grin

LangChain4j

190

190

LangChain4j - liens officiels (1/3)

- <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 : <https://github.com/langchain4j/langchain4j>
- Différentes versions et leurs différences : <https://github.com/langchain4j/langchain4j/releases>
- Problèmes (issues) répertoriés avec LangChain4j : <https://github.com/langchain4j/langchain4j/issues>

R. Grin

LangChain4j

191

191

LangChain4j - liens officiels (2/3)

- 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>
- Get started : <https://docs.langchain4j.dev/get-started>
- Discussions : <https://github.com/langchain4j/langchain4j/discussions>
- Q&A : <https://github.com/langchain4j/langchain4j/discussions/categories/q-a>

R. Grin

LangChain4j

192

192

LangChain4j - liens officiels (3/3)

- Intégration Gemini :
<https://github.com/langchain4j/langchain4j/blob/main/docs/docs/integrations/language-models/google-ai-gemini.md>
- Intégration OpenAI :
<https://github.com/langchain4j/langchain4j/blob/main/docs/docs/integrations/language-models/open-ai-official.md>

R. Grin

LangChain4j

193

193

Anciennes versions de LangChain4j

- Pour la javadoc, aller sur <https://javadoc.io/doc/dev.langchain4j/langchain4j-core> et choisir la version en haut
- Pour le reste, aller sur le dépôt Maven Central ; par exemple <https://mvnrepository.com/artifact/dev.langchain4j/langchain4j/0.36.0> ; en cliquant sur « View All » de la ligne Files, on peut récupérer, par exemple, la javadoc ou le source

R. Grin

LangChain4j

194

194

Code source

- Le code est constitué de modules
 - langchain4j est le module principal
 - langchain4j-core contient les classes de base : Document, TextSegment, ...
 - de nombreux modules dédiés aux différents produits supportés par LangChain4j : langchain4j-openai, langchain4j-google-ai-gemini, langchain4j-hugging-face, ...
- Lombok était utilisé au début de LangChain4j mais il en cours de suppression ; du code ancien utilise encore Lombok et des getters et setters n'apparaissent alors ni dans le code, ni dans la javadoc

R. Grin

LangChain4j

195

195

LangChain4j (1/3)

- https://www.youtube.com/watch?v=cjI_6Siry-s
- <https://www.youtube.com/watch?v=EwriKYptLao>
- 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/>

R. Grin

LangChain4j

196

196

LangChain4j (2/3)

- 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

R. Grin

LangChain4j

197

197

LangChain4j (3/3)

- <https://kindgeek.com/blog/post/experiments-with-langchain4j-or-java-way-to-llm-powered-applications> : article très complet sur LangChain4j (6/2/24)
- Vidéo complète sur les outils (@Tool) : https://youtu.be/cjI_6Siry-s?si=nAk9AjK2dajT2b63
- Vidéo de presque 3 heures sur LangChain4j : <https://www.youtube.com/watch?v=jzuP6l54kWA>
- <https://javapro.io/2025/04/23/build-ai-apps-and-agents-in-java-hands-on-with-langchain4j/>

R. Grin

LangChain4j

198

198

LangChain4j et réseaux sociaux

- Stackoverflow :
<https://stackoverflow.com/questions/tagged/langchain4j>
- Discord :
<https://discord.com/channels/1156626270772269217/1156626271212666882>

R. Grin

LangChain4j

199

199

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/>)

R. Grin

LangChain4j

200

200

LangChain4j et LMs

- Particularité des LMs :
<https://docs.langchain4j.dev/category/language-models/>
- Pour Gemini :
<https://docs.langchain4j.dev/integrations/language-models/google-ai-gemini>
- Pour OpenAI :
<https://docs.langchain4j.dev/integrations/language-models/open-ai>

R. Grin

LangChain4j

201

201

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

R. Grin

LangChain4j

202

202

MOOCs et vidéos sur LangChain

- La playlist de courtes vidéos sur les LMs, ChatGPT, LangChain :
https://www.youtube.com/playlist?list=PLKW0AUxdZEb_BqGgm-Rk7fiPUXccFHBGB
- <https://www.youtube.com/watch?v=uJJ6uP5lViA>
- <https://www.youtube.com/watch?v=1VeYoNoXlGU>

R. Grin

LangChain4j

203

203

- 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=RAubwMSPRT0>
 - 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/>

R. Grin

LangChain4j

204

204



Projets avec LangChain4j

- <https://github.com/langchain4j/awesome-langchain4j>

R. Grin LangChain4j 205

205