LangChain4j

**Types of ChatMessages**

There are currently four types of chat messages, one for each "source" of the message:

* UserMessage: This is a message from the user. The user can be either an end user of your application (a human) or your application itself. Depending on the modalities supported by the LLM, UserMessage can contain either just text (String), or text and/or images (Image).
* AiMessage: This is a message that was generated by the AI, usually in response to the UserMessage. As you might have noted, the generate method returns an AiMessage wrapped in a Response. AiMessage can contain either a textual response (String), or a request to execute a tool (ToolExecutionRequest). Don't worry, we will explore tools a bit later.
* ToolExecutionResultMessage: This is the result of the ToolExecutionRequest. We will cover this more a bit later.
* SystemMessage: This is a message from the system. Usually, you, as a developer, should define the content of this message. Normally, you would write here instructions on what the LLM's role is in this conversation, how it should behave, in what style to answer, etc. LLMs are trained to pay more attention to SystemMessage than to other types of messages, so be careful and it's better not to give an end user free access to define or inject some input into a SystemMessage. Usually, it is located at the start of the conversation.

**Chat Memory**

Maintaining and managing ChatMessages manually is cumbersome. Therefore, ChatMemory exists. It essentially acts as a container for ChatMessages (backed by a List), with additional features like persistence (see ChatMemoryStore) and mechanisms such as an "eviction policy." An eviction policy is necessary for two main reasons:

* LLMs have a limited context window, meaning there's a cap on the number of tokens they can process at any given time. Eventually, the entire conversation might exceed this limit.
* Each token has a cost, making each call to the LLM progressively more expensive.

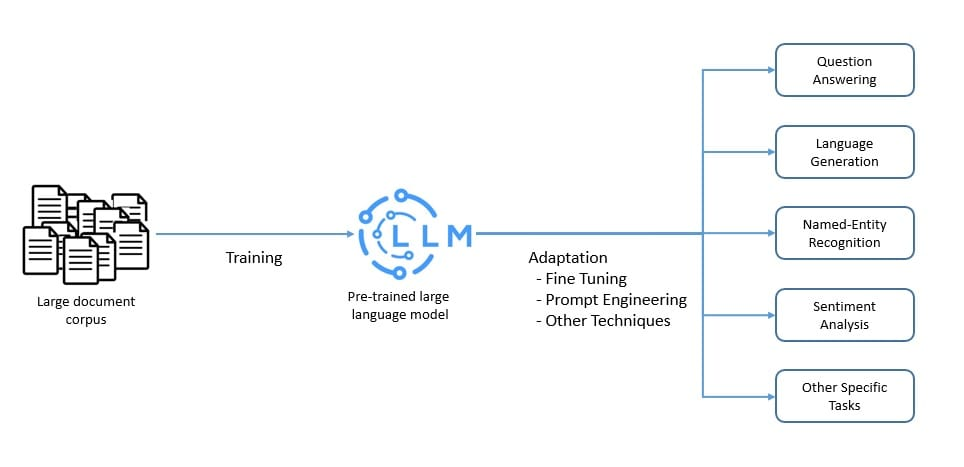
Currently, LangChain4j implements two algorithms for eviction policy:

* The simpler one, MessageWindowChatMemory, functions as a sliding window, retaining the N most recent messages and evicting older ones that no longer fit. SystemMessage is an exception; it is never evicted. However, because each message can contain a varying number of tokens, MessageWindowChatMemory is mostly useful for fast prototyping.
* A more sophisticated option is the TokenWindowChatMemory, which also operates as a sliding window but focuses on keeping the N most recent **tokens**, evicting older messages as needed. It requires a Tokenizer to count the tokens in each ChatMessage. Like before, SystemMessage is always preserved.

| **Parameter** | **Description** | **Type** |
| --- | --- | --- |
| modelName | The name of the model to use (gtp-3.5-turbo, gpt-4-1106-preview, ...) | String |
| temperature | What sampling temperature to use, between 0 and 2. Higher values like 0.8 will make the output more random, while lower values like 0.2 will make it more focused and deterministic. | Double |
| max\_tokens | The maximum number of tokens that can be generated in the chat completion. | Integer |
| frequencyPenalty | Number between -2.0 and 2.0. Positive values penalize new tokens based on their existing frequency in the text so far, decreasing the model's likelihood to repeat the same line verbatim. | Double |
| ... | ... | ... |

SystemMessage Generation

You provide the Class of your interface to AiServices along with the low-level components, and AiServices creates a proxy object implementing this interface. Currently, it uses reflection, but we are considering alternatives as well. This proxy object handles all the conversions for inputs and outputs. In this case, the input is a single String, but we are using a ChatLanguageModel which takes ChatMessage as input. So, AiService will automatically convert it into a UserMessage and invoke ChatLanguageModel. Since the output type of the chat method is a String, after ChatLanguageModel returns AiMessage, it will be converted into a String before being returned from the chat method.



JSON mode

When extracting custom POJOs (actually JSON, which is then parsed into the POJO), it is recommended to enable a "JSON mode" in the model configuration. This way, the LLM will be forced to produce valid JSON.

For OpenAI:

OpenAiChatModel.builder()

...

.responseFormat("json\_object")

.build();