响应流

> **笔记**
>
> 本页介绍如何使用低级 LLM API 实现响应流式传输。有关高级 LLM API，请参阅AI 服务。

LLM 每次生成一个 token，因此许多 LLM 提供商都提供逐个 token 流式传输响应的方式，而无需等待整个文本生成完毕。这显著提升了用户体验，因为用户无需等待未知的时间，几乎可以立即开始阅读响应。

对于`ChatModel`和`LanguageModel`接口，有对应的`StreamingChatModel`和`StreamingLanguageModel`接口。它们具有类似的 API，但可以流式传输响应。它们接受`StreamingChatResponseHandler`接口的实现作为参数。

```
public interface StreamingChatResponseHandler {

    void onPartialResponse(String partialResponse);

    default void onPartialThinking(PartialThinking partialThinking) {}

    default void onPartialToolCall(PartialToolCall partialToolCall) {}

    default void onCompleteToolCall(CompleteToolCall completeToolCall) {}

    void onCompleteResponse(ChatResponse completeResponse);

    void onError(Throwable error);
}
```

通过实施`StreamingChatResponseHandler`，您可以为以下事件定义操作：

- 生成下一个部分文本响应时：`onPartialResponse(String)`被调用。根据 LLM 提供程序的不同，部分响应文本可以包含一个或多个令牌。例如，您可以在令牌可用时立即将其直接发送到 UI。
- 生成下一个部分思考/推理文本时：`onPartialThinking(PartialThinking)`被调用。根据 LLM 提供商的不同，部分思考文本可以由一个或多个标记组成。
- 当生成下一个部分工具调用`onPartialToolCall(PartialToolCall)`时：被调用。

- 当 LLM 完成单个工具调用的流式传输时：`onCompleteToolCall(CompleteToolCall)`被调用。
- 当 LLM 完成生成时：`onCompleteResponse(ChatResponse)`将被调用。该`ChatResponse`对象包含完整的响应（`AiMessage`）以及`ChatResponseMetadata`。
- 当发生错误时：`onError(Throwable error)`被调用。

下面是如何使用实现流式传输的示例`StreamingChatModel`：

```
StreamingChatModel model = OpenAiStreamingChatModel.builder()
    .apiKey(System.getenv("OPENAI_API_KEY"))
    .modelName(GPT_4_O_MINI)
    .build();

String userMessage = "Tell me a joke";

model.chat(userMessage, new StreamingChatResponseHandler() {

    @Override
    public void onPartialResponse(String partialResponse) {
        System.out.println("onPartialResponse: " + partialResponse);
    }

    @Override
    public void onPartialThinking(PartialThinking partialThinking) {
        System.out.println("onPartialThinking: " + partialThinking);
    }

    @Override
    public void onPartialToolCall(PartialToolCall partialToolCall) {
        System.out.println("onPartialToolCall: " + partialToolCall);
    }

    @Override
    public void onCompleteToolCall(CompleteToolCall completeToolCall) {
        System.out.println("onCompleteToolCall: " + completeToolCall);
    }

    @Override
    public void onCompleteResponse(ChatResponse completeResponse) {
        System.out.println("onCompleteResponse: " + completeResponse);
    }

    @Override
    public void onError(Throwable error) {
        error.printStackTrace();
```

```
    }
});
```

一种更简洁的流式传输响应的方法是使用`LambdaStreamingResponseHandler`类。这个实用程序类提供了静态方法来创建`StreamingChatResponseHandler`使用 Lambda 表达式的流式传输。使用 Lambda 表达式来流式传输响应的方法非常简单。只需使用`onPartialResponse()`定义如何处理部分响应的 Lambda 表达式调用静态方法即可：

```
import static
dev.langchain4j.model.LambdaStreamingResponseHandler.onPartialResponse;

model.chat("Tell me a joke", onPartialResponse(System.out::print));
```

该方法允许您为和事件`onPartialResponseAndError()`定义操作：
`onPartialResponse()` `onError()`

```
import static
dev.langchain4j.model.LambdaStreamingResponseHandler.onPartialResponseAndError;

model.chat("Tell me a joke", onPartialResponseAndError(System.out::print,
Throwable::printStackTrace));
```

编辑此页面