Interface | Langchain

Skip to main content LangChainDocsUse casesIntegrationsAPIMoreCommunityTutorialsContributingAlso by LangChainChat our docsLangSmithLangChain HubLangServePython DocsSearchCTRLKGet startedIntroductionInstallationQuickstartLangChain Expression LanguageInterfaceHow toCookbookWhy use LCEL?LangChain Expression Language (LCEL)ModulesModel I/ORetrievalChainsMemoryAgentsCallbacksModulesSecurityGuidesEcosystemLangChain Expression LanguageInterfaceOn this pageInterface In an effort to make it as easy as possible to create custom chains, we've implemented a "Runnable" protocol that most components implement.

This is a standard interface with a few different methods, which make it easy to define custom chains as well as making it possible to invoke them in a standard way. The standard interface exposed includes:stream: stream back chunks of the responseinvoke: call the chain on an inputbatch: call the chain on a list of inputsThe input type varies by component :ComponentInput TypePromptObjectRetrieverSingle stringLLM, ChatModelSingle string, list of chat messages or

PromptValueToolSingle string, or object, depending on the toolOutputParserThe output of an LLM or ChatModelThe output type also varies by component :ComponentOutput TypeLLMStringChatModelChatMessagePromptPromptValueRetrieverList of documentsToolDepends on the toolOutputParserDepends on the parserYou can combine runnables (and runnable-like objects such as functions and objects whose values are all functions) into sequences in two ways:Call the .pipe instance method, which takes another runnable-like as an argumentUse the RunnableSequence.from([]) static method with an array of runnable-likes, which will run in sequence when invokedSee below for examples of how this looks.Streamimport { PromptTemplate } from "langchain/prompts";import { ChatOpenAI } from "langchain/chat_models/openai";const model = new ChatOpenAI({});const promptTemplate = PromptTemplate.fromTemplate("Tell me a joke about {topic}");const chain = promptTemplate.pipe(model);const stream = await chain.stream({ topic: "bears" });// Each chunk has the same interface as a chat messagefor await (const chunk of stream) { console.log(chunk?.content);}/*Why don't bears wear shoes?Because they have bear feet!*/API Reference:PromptTemplate from langchain/promptsChatOpenAI from langchain/chat_models/openaiInvokeimport { PromptTemplate } from "langchain/prompts";import { ChatOpenAI } from "langchain/chat_models/openai";import { RunnableSequence } from "langchain/schema/runnable";const model = new ChatOpenAI({});const promptTemplate = PromptTemplate.fromTemplate("Tell me a joke about {topic}");// You can also create a chain using an array of runnablesconst chain = RunnableSequence.from([promptTemplate, model]);const result = await chain.invoke({ topic: "bears" });console.log(result);/* AIMessage { content: "Why don't bears wear shoes?\n\nBecause they have bear feet!", }*/API Reference:PromptTemplate from langchain/promptsChatOpenAI from langchain/chat_models/openaiRunnableSequence from langchain/schema/runnableBatchimport { PromptTemplate } from "langchain/prompts";import { ChatOpenAI } from "langchain/chat_models/openai";const model = new ChatOpenAI({});const promptTemplate = PromptTemplate.fromTemplate("Tell me a joke about {topic}");const chain = promptTemplate.pipe(model);const result = await chain.batch([{ topic: "bears" }, { topic: "cats" }]);console.log(result);/* [AIMessage {   content: "Why don't bears wear shoes?\n\nBecause they have bear feet!",   },   AIMessage {   content: "Why don't cats play poker in the wild?\n\nToo many cheetahs!"   }]*/API Reference:PromptTemplate from langchain/promptsChatOpenAI from langchain/chat_models/openaiYou can also pass a batchOptions argument to the call. There are options to set maximum concurrency

and whether or not to return exceptions instead of throwing them (useful for gracefully handling failures!):import { PromptTemplate } from "langchain/prompts";import { ChatOpenAI } from "langchain/chat_models/openai";const model = new ChatOpenAI({ modelName: "badmodel",});const promptTemplate = PromptTemplate.fromTemplate("Tell me a joke about {topic}");const chain = promptTemplate.pipe(model);const result = await chain.batch([{ topic: "bears" }, { topic: "cats" }], {}, { returnExceptions: true, maxConcurrency: 1 });console.log(result);/* [NotFoundError: The model `badmodel` does not exist   at Function.generate (/Users/jacoblee/langchain/langchainjs/node_modules/openai/src/error.ts:71:6)   at OpenAI.makeStatusError (/Users/jacoblee/langchain/langchainjs/node_modules/openai/src/core.ts:381:13)   at OpenAI.makeRequest (/Users/jacoblee/langchain/langchainjs/node_modules/openai/src/core.ts:442:15)   at process.processTicksAndRejections (node:internal/process/task_queues:95:5)   at async file:///Users/jacoblee/langchain/langchainjs/langchain/dist/chat_models/openai.js:514:29   at RetryOperation._fn (/Users/jacoblee/langchain/langchainjs/node_modules/p-retry/index.js:50:12) {   status: 404,   NotFoundError: The model `badmodel` does not exist   at Function.generate (/Users/jacoblee/langchain/langchainjs/node_modules/openai/src/error.ts:71:6)   at OpenAI.makeStatusError (/Users/jacoblee/langchain/langchainjs/node_modules/openai/src/core.ts:381:13)   at

OpenAI.makeRequest (/Users/jacoblee/langchain/langchainjs/node_modules/openai/src/core.ts:442:15) at process.processTicksAndRejections (node:internal/process/task_queues:95:5) at async file:///Users/jacoblee/langchain/langchainjs/langchain/dist/chat_models/openai.js:514:29 at RetryOperation._fn (/Users/jacoblee/langchain/langchainjs/node_modules/p-retry/index.js:50:12) {

status: 404,]*/API Reference:PromptTemplate from langchain/promptsChatOpenAI from langchain/chat_models/openaiPreviousLangChain Expression Language (LCEL)NextRoute between multiple runnablesStreamInvokeBatchCommunityDiscordTwitterGitHubPythonJS/TSMoreHomepageBlogCopyright © 2023 LangChain, Inc.