

12 | 深度剖析（上）：Ling框架的底层实现

月影 · 跟月影学前端智能体开发



你好，我是月影。首先给假期还在学习的你点个赞。

上节课，我们已经通过实践初步了解了 Ling 的使用方法，那么接下来两节课，我们就来深入 Ling 的实现细节，看看 Ling 框架里几个模块的具体设计，以及它们是如何协同工作的。掌握了这些，也会对你去处理复杂的 AI 工作流有所启发。

Ling 的四个子系统

Ling 框架包含四个子系统。

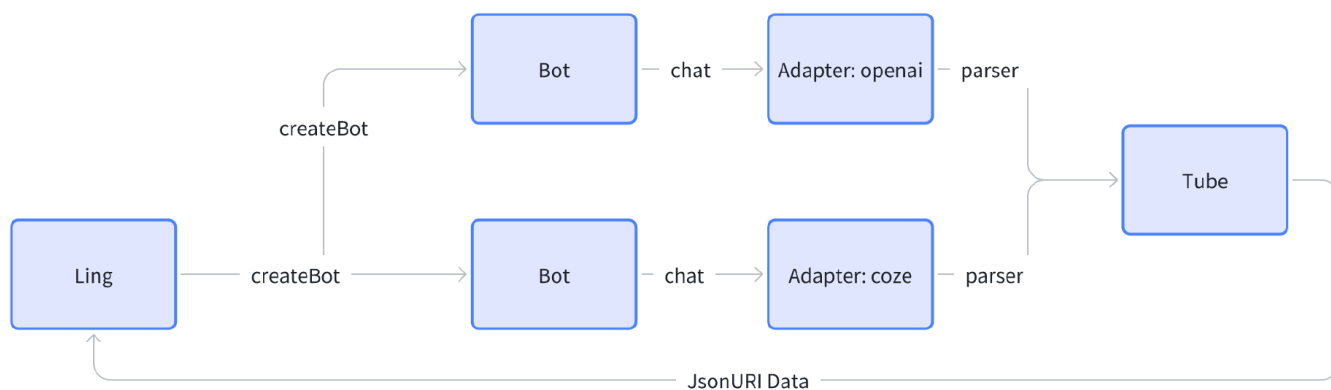
adapter：大模型 API 底层模块适配器，目前支持标准 OpenAI 和 Coze 两类文本大模型 API。

bot：对大模型节点的抽象，负责管理和控制单一节点。

parser：JSONParser 实现，这是一个可独立使用的子系统，前面课程中已经使用过。

tube：对流式（Streaming）对象的封装、前后端通讯的数据格式定义以及事件管理。

在使用 Ling 框架的时候，我们通过创建 Ling 对象实例来管理 bot。bot 在内部处理节点输入输出时调用 adapter，根据配置的模型参数，由 adapter 选择具体的 API 调用。在 adapter 具体调用 API 过程中会通过 parser 来动态解析大模型输入输出，并将处理好的数据通过 tube 发送，最后再由 tube 转发给前端。



下面我们来分别深入拆解一下这些模块，由于 Parser 模块就是动态 JSON 解析模块，在前面的章节中我们已经单独拆解过了，因此接下来我们主要分别介绍一下其他三个模块以及最外层的 Ling 管理模块。

在这一节课，我们主要关注 Adapter 模块，因为它是真正执行大模型对话的模块，因此是**底层中的底层模块**。

Adapter 子模块

Adapter 模块是暴露给 Bot 的调用大模型对话接口，它的目的是兼容各种不同的 API 规格，目前只默认兼容 OpenAI 和 Coze 两种规格，由于大部分文本大模型都兼容 OpenAI 规格，所以这样已经基本上覆盖了几乎所有常用的文本大模型。

我们分别看一下这两种规格的具体实现代码。

OpenAI 封装

首先是 OpenAI：

```
1 import OpenAI from 'openai';
2
3 import { AzureOpenAI } from "openai";
4 import "@azure/openai/types";
5
6 import { ChatConfig, ChatOptions } from '../..types';
7 import { Tube } from '../..tube';
8 import { JSONParser } from '../..parser';
9 import { sleep } from '../..utils';
10
11 import "dotenv/config";
12
13 const DEFAULT_CHAT_OPTIONS = {
14   temperature: 0.9,
15   top_p: 1,
16   frequency_penalty: 0,
17   presence_penalty: 0,
18 };
19
20 export async function getChatCompletions(
21   tube: Tube,
22   messages: any[],
23   config: ChatConfig,
24   options?: ChatOptions,
25   onComplete?: (content: string) => void,
26   onStringResponse?: (content: {uri: string|null, delta: string} | string) => voi
27   onObjectResponse?: (content: {uri: string|null, delta: any}) => void
28 ) {
29   options = {...DEFAULT_CHAT_OPTIONS, ...options};
30   if (options.response_format) { // 防止原始引用对象里的值被删除
31     options.response_format = {type: options.response_format.type, root: options.
32   }
33   options.max_tokens = options.max_tokens || config.max_tokens || 4096; // || 163
34
35   const isQuiet: boolean = !!options.quiet;
36   delete options.quiet;
37
38   const isJSONFormat = options.response_format?.type === 'json_object';
39
40   let client: OpenAI | AzureOpenAI;
41   let model = '';
42   if(config.endpoint.endsWith('openai.azure.com')) {
43     process.env.AZURE_OPENAI_ENDPOINT=config.endpoint;
44     const scope = "https://cognitiveservices.azure.com/.default";
45     const deployment = config.model_name;
46     const apiVersion = config.api_version || "2024-07-01-preview";
47     client = new AzureOpenAI({
```


```
48     endpoint: config.endpoint,
49     apiKey: config.api_key,
50     apiVersion,
51     deployment });
52 } else {
53     const {model_name, api_key, endpoint} = config as ChatConfig;
54     model = model_name;
55     client = new OpenAI({
56         apiKey: api_key,
57         baseUrl: endpoint.replace(/\/chat\/completions$/, ''),
58         dangerouslyAllowBrowser: true,
59     });
60 }
61
62 const parentPath = options.response_format?.root;
63 delete options.response_format.root;
64
65 const events = await client.chat.completions.create({
66     messages,
67     ...options,
68     model,
69     stream: true,
70 });
71
72 let content = '';
73 const buffer: any[] = [];
74 let done = false;
75
76 let parser: JSONParser | undefined;
77
78 if (isJSONFormat) {
79     parser = new JSONParser({
80         parentPath,
81         autoFix: true,
82     });
83     parser.on('data', (data) => {
84         buffer.push(data);
85     });
86     parser.on('string-resolve', (content) => {
87         if (onStringResponse) onStringResponse(content);
88     });
89     parser.on('object-resolve', (content) => {
90         if (onObjectResponse) onObjectResponse(content);
91     });
92 }
93
94 const promises: any[] = [
95     (async () => {
96         for await (const event of events) {
```

```

97     if (tube.canceled) break;
98     const choice = event.choices[0];
99     if (choice && choice.delta) {
100         if (choice.delta.content) {
101             content += choice.delta.content;
102             if (parser) { // JSON format
103                 parser.trace(choice.delta.content);
104             } else {
105                 buffer.push({ uri: parentPath, delta: choice.delta.content });
106             }
107         }
108     }
109 }
110 done = true;
111 if (parser) {
112     parser.finish();
113 }
114 })(),
115 (async () => {
116     let i = 0;
117     while (!(done && i >= buffer.length)) {
118         if (i < buffer.length) {
119             tube.enqueue(buffer[i], isQuiet);
120             i++;
121         }
122         const delta = buffer.length - i;
123         if (done || delta <= 0) await sleep(10);
124         else await sleep(Math.max(10, 1000 / delta));
125     }
126     if (!tube.canceled && onComplete) onComplete(content);
127 })(),
128 ];
129 await Promise.race(promises);
130 if (!isJSONFormat && onStringResponse) onStringResponse({ uri: parentPath, delt
131 return content; // inference done
132 }

```

我们先整体扫一下上面的代码，你会发现实际上它兼容两种子规格，分别是标准的 OpenAI 和微软的 AzureOpenAI，这两种规格除了对象创建不同外，其他的部分是完全兼容的。所以我们只需要根据不同的 config.endpoint 判断，如果是 openai.azure.com，那么就通过 AzureOpenAI 类创建对象，否则就通过 OpenAI 类创建对象。

 复制代码

```

1     if(config.endpoint.endsWith('openai.azure.com')) {

```

```

2    process.env.AZURE_OPENAI_ENDPOINT=config.endpoint;
3    const scope = "https://cognitiveservices.azure.com/.default";
4    const deployment = config.model_name;
5    const apiVersion = config.api_version || "2024-07-01-preview";
6    client = new AzureOpenAI({
7        endpoint: config.endpoint,
8        apiKey: config.api_key,
9        apiVersion,
10       deployment });
11 } else {
12     const {model_name, api_key, endpoint} = config as ChatConfig;
13     model = model_name;
14     client = new OpenAI({
15         apiKey: api_key,
16         baseURL: endpoint.replace(/\/chat\/completions$/, ''),
17         dangerouslyAllowBrowser: true,
18     });
19 }

```

注意模块暴露的接口是 `getChatCompletions` 函数，它有较多的参数，我们分别来看一下。

`tube`: Tube 子模块对象，负责接收推理输出的流式信息。

`messages`: 对话消息，一般是标准的大模型消息数组，每一条大模型消息由`{role: user|assistant|system, content: string}` 构成。

`config`: `ChatConfig` 对象，大模型基础配置信息，详见下方类型定义。


`options`: `ChatOptions` 对象，大模型对话配置信息，详见下方类型定义。

`onComplete`: 对话结束的回调函数。

`onStringResponse`: `parser` 解析内容中触发 `onStringResolved` 时调用该回调函数。

`onObjectResponse` : `parser` 解析内容中触发 `onObjectResolved` 时调用该回调函数。

以下是 `ChatConfig` 对象和 `ChatOptions` 对象定义：

 复制代码

```

1 export interface ChatConfig {
2     model_name: string;
3     endpoint: string;
4     api_key: string;

```

```
5  api_version?: string;
6  session_id?: string;
7  max_tokens?: number;
8  sse?: boolean;
9  }
10
11 export interface ChatOptions {
12   temperature?: number;
13   presence_penalty?: number;
14   frequency_penalty?: number;
15   stop?: string[];
16   top_p?: number;
17   response_format?: any;
18   max_tokens?: number;
19   quiet?: boolean;
20 }
```

ChatConfig 主要是选择大模型的配置，包括必选模型名 `model_name`（Coze 则设置为 `coze:${botId}`）、服务地址 `endpoint`、鉴权信息 `api_key` 以及可选的 `api_version`（主要给 azure 平台使用）、`session_id`（Coze 模型需要作为 `user_id`）、`max_tokens`（一次对话最多 token 数量）、`sse`（是否采用 Server-Sent Events）。

ChatOptions 则主要是大模型推理输出内容的配置，包括以下参数：

temperature：用来控制生成文本时的“随机性”或“创造性”，数值范围通常在 0 ~ 2 之间，或具体根据模型而定。越接近 0，输出越“确定”；越大，输出越“多样”或“发散”。

presence_penalty：出现惩罚（presence penalty）用来惩罚已经出现过的 token（词或词片段），从而鼓励模型输出更多的新信息，减少重复。数值越高，模型越倾向于避免重复先前已经生成的内容。

frequency_penalty：频率惩罚（frequency penalty）与 presence penalty 类似，也是一种惩罚机制，主要作用是在生成文本的过程中对于频繁出现的 token 进行惩罚，减少重复。与 presence penalty 不同之处在于，它根据某一 token 在整个对话或上下文中出现的“频率”来做惩罚。一般来说，presence_penalty 是只要出现过某 token，就会增加惩罚，而 frequency_penalty 则是基于其出现次数来累积惩罚。

stop: 停止词 (stop tokens) 。在模型生成文本时，如果遇到这里定义的任意一个字符串，就会停止继续生成。用于控制回复的长度或终止条件，防止输出过长或产生不需要的内容。


top_p : 核采样 (top-p) 参数，与 temperature 类似，也是控制采样随机性或多样性的手段之一。在核采样中，模型会从累积概率达到 top-p 阈值的词汇中进行抽样，而不是从所有词汇里进行挑选。数值范围在 0 ~ 1 之间。数值越小，模型从概率最高的一部分候选词中采样（输出将更保守）；数值越大，候选词会更多，输出更具多样性。

response_format: 指定回复的格式。DeepSeek、Kimi 支持强制以 JSON 格式回复，这样模型会保证生成内容 JSON 语法的正确性。

max_tokens: 生成文本的最大长度限制，即最多能生成多少个 token（词或词片段）。


quiet: 是否静默输出，如果将 quiet 设为 true，那么流式输出的过程中，内容将不会被转发给 Tube 对象，也就不会被默认发送给客户端。

接着通过 OpenAI SDK 调用对话主体部分：

 复制代码

```
1 const events = await client.chat.completions.create({
2   messages,
3   ...options,
4   model,
5   stream: true,
6 });
```

我们要通过 response_type 判断一下输出格式是否是 JSON，如果是 JSON 格式，那么我们采用 JSONParser 进行解析，将解析内容存入缓冲区。

 复制代码

```
1 if (isJSONFormat) {
2   parser = new JSONParser({
3     parentPath,
4     autoFix: true,
5   });
6   parser.on('data', (data) => {
7     buffer.push(data);
```




```

8     });
9     parser.on('string-resolve', (content) => {
10         if (onStringResponse) onStringResponse(content);
11     });
12     parser.on('object-resolve', (content) => {
13         if (onObjectResponse) onObjectResponse(content);
14     });
15 }

```

再之后是模块主体逻辑的核心部分，通过遍历 events 来解析内容进行处理：

 复制代码

```


1  const promises: any[] = [
2      (async () => {
3          for await (const event of events) {
4              if (tube.canceled) break;
5              const choice = event.choices[0];
6              if (choice && choice.delta) {
7                  if (choice.delta.content) {
8                      content += choice.delta.content;
9                      if (parser) { // JSON format
10                         parser.trace(choice.delta.content);
11                     } else {
12                         buffer.push({ uri: parentPath, delta: choice.delta.content });
13                     }
14                 }
15             }
16         }
17         done = true;
18         if (parser) {
19             parser.finish();
20         }
21     })(),
22     (async () => {
23         let i = 0;
24         while (!(done && i >= buffer.length)) {
25             if (i < buffer.length) {
26                 tube.enqueue(buffer[i], isQuiet);
27                 i++;
28             }
29             const delta = buffer.length - i;
30             if (done || delta <= 0) await sleep(10);
31             else await sleep(Math.max(10, 1000 / delta));
32         }
33         if (!tube.canceled && onComplete) onComplete(content);

```

```
34     })(),
35   ];
36   await Promise.race(promises);
```

注意这里我们有一个小设计，我们先处理内容放入缓冲对象 **buffer**，然后再用一个异步过程，将对象中的内容一一放入 **Tube** 对象的流式队列里，这样两步分离能够避免一次发送太多内容导致的阻塞。

最后我们解析完成后，即可返回 **content** 内容。如果是非 JSON 格式，那么我们在内容生成完毕后，补一个 **onStringResponse** 回调。


 复制代码

```
1   if (!isJSONFormat && onStringResponse) onStringResponse({ uri: parentPath, delt
2   return content; // inference done
```

这样我们就实现了兼容 OpenAI 的 API 的调用逻辑主体部分。

Coze 封装

如果 **ChatConfig** 的 **model_name** 以 **coze:** 开头，那么由 Bot 会选择调用 **Adapter** 的 **Coze** 封装，我们来看一下 **Coze** 封装的具体实现代码：

 复制代码

```
1 import { ChatConfig, ChatOptions } from '../..//types';
2 import { Tube } from '../..//tube';
3 import { JSONParser } from '../..//parser';
4 import { sleep } from '../..//utils';
5
6 export async function getChatCompletions(
7   tube: Tube,
8   messages: any[],
9   config: ChatConfig,
10  options?: ChatOptions & {custom_variables?: Record<string, string>},
11  onComplete?: (content: string, function_calls?: any[]) => void,
12  onStringResponse?: (content: {uri: string|null, delta: string} | string) => voi
13  onObjectResponse?: (content: {uri: string|null, delta: any}) => void
14 ) {
15   const bot_id = config.model_name.split(':')[1]; // coze:bot_id
```

```
16  const { api_key, endpoint } = config as ChatConfig;
17
18  const isQuiet: boolean = !!options?.quiet;
19  delete options?.quiet;
20
21  const isJSONFormat = options?.response_format?.type === 'json_object';
22
23  // system
24  let system = '';
25  const systemPrompts = messages.filter((message) => message.role === 'system');
26  if (systemPrompts.length > 0) {
27    system = systemPrompts.map((message) => message.content).join('\n\n');
28    messages = messages.filter((message) => message.role !== system);
29  }
30
31  const custom_variables = { systemPrompt: system, ...options?.custom_variables }
32  const query = messages.pop();
33
34
35  let chat_history = messages.map((message) => {
36    if (message.role === 'function') {
37      return {
38        role: 'assistant',
39        type: 'tool_response',
40        content: message.content,
41        content_type: 'text',
42      };
43    } else if (message.role === 'assistant' && message.function_call) {
44      return {
45        role: 'assistant',
46        type: 'function_call',
47        content: JSON.stringify(message.function_call),
48        content_type: 'text',
49      };
50    } else if (message.role === 'assistant') {
51      return {
52        role: 'assistant',
53        type: 'answer',
54        content: message.content,
55        content_type: 'text',
56      };
57    }
58    return {
59      role: message.role,
60      content: message.content,
61      content_type: 'text',
62    };
63  });
64
```

```
65 let parser: JSONParser | undefined;
66 const parentPath = options?.response_format?.root;
67
68 if (isJSONFormat) {
69     parser = new JSONParser({
70         parentPath,
71         autoFix: true,
72     });
73     parser.on('data', (data) => {
74         tube.enqueue(data, isQuiet);
75     });
76     parser.on('string-resolve', (content) => {
77         if (onStringResponse) onStringResponse(content);
78     });
79     parser.on('object-resolve', (content) => {
80         if (onObjectResponse) onObjectResponse(content);
81     });
82 }
83
84 const _payload = {
85     bot_id,
86     user: 'bearbobo',
87     query: query.content,
88     chat_history,
89     stream: true,
90     custom_variables,
91 } as any;
92
93 const body = JSON.stringify(_payload, null, 2);
94
95 const res = await fetch(endpoint, {
96     method: 'POST',
97     headers: {
98         'Content-Type': 'application/json',
99         Authorization: `Bearer ${api_key}`,
100     },
101     body,
102 });
103
104 const reader = res.body?.getReader();
105 if(!reader) {
106     console.error('No reader');
107     tube.cancel();
108     return;
109 }
110
111 let content = '';
112 const enc = new TextDecoder('utf-8');
113 let buffer = '';
```

```
114 let functionCalling = false;
115 const function_calls = [];
116 let funcName = '';
117
118 do {
119     if (tube.canceled) break;
120     const { done, value } = await reader.read();
121     if(done) break;
122     const delta = enc.decode(value);
123     const events = delta.split('\n\n');
124     for (const event of events) {
125         // console.log('event', event);
126         if (/^\s*data:/.test(event)) {
127             buffer += event.replace(/^\s*data:\s*/, '');
128             let data;
129             try {
130                 data = JSON.parse(buffer);
131             } catch (ex) {
132                 console.error(ex, buffer);
133                 continue;
134             }
135             buffer = '';
136             if (data.error_information) {
137                 // console.error(data.error_information.err_msg);
138                 tube.enqueue({event: 'error', data});
139                 tube.cancel();
140                 break;
141             }
142             const message = data.message;
143             if (message) {
144                 if (message.type === 'answer') {
145                     let result = message.content;
146                     if(!content) { // 去掉开头的空格, coze有时候会出现这种情况, 会影响 markdown
147                         result = result.trimStart();
148                     }
149                     if(!result) continue;
150                     content += result;
151                     const chars = [...result];
152                     for (let i = 0; i < chars.length; i++) {
153                         if (parser) {
154                             parser.trace(chars[i]);
155                         } else {
156                             tube.enqueue({ uri: parentPath, delta: chars[i] }, isQuiet);
157                         }
158                         await sleep(50);
159                     }
160                 }
161             }
162             } else {
```

```

163         try {
164             const data = JSON.parse(event);
165             if (data.code) {
166                 tube.enqueue({event: 'error', data});
167                 tube.cancel();
168             }
169         } catch(ex) {}
170     }
171 }
172 } while (1);
173 if (!isJSONFormat && onStringResponse) onStringResponse({ uri: parentPath, delt
174 if (!tube.canceled && onComplete) onComplete(content, function_calls);
175 return content;
176 }

```


这里一些格式转换的细节我就不说了，大家可以自己去看代码。这里就说一些关键部分。

与 OpenAI 封装不同的是，OpenAI 调用 OpenAI sdk 提供的方法来完成对话，而 Coze 则直接使用 HTTP 请求：

```

1  const res = await fetch(endpoint, {
2      method: 'POST',
3      headers: {
4          'Content-Type': 'application/json',
5          Authorization: `Bearer ${api_key}`,
6      },
7      body,
8  });

```

 复制代码

然后我们用调用 `reader.read()` 的方式读出流式内容，进行处理，处理逻辑和 OpenAI 类似，我们在前面课程中也介绍过流式输出的几个例子，实现方法都差不多，大家自己熟悉一下就可以了。

唯一要注意的是，Coze 中的 message 当 role 是 assistant 时，有个额外的 type 属性，一般设置为 answer 就可以了。

```
1 if (message.type === 'answer') {
2   let result = message.content;
3   if(!content) { // 去掉开头的空格, coze有时候会出现这种情况, 会影响 markdown 格式
4     result = result.trimStart();
5     if(!result) continue;
6   }
7   content += result;
8   const chars = [...result];
9   for (let i = 0; i < chars.length; i++) {
10    if (parser) {
11      parser.trace(chars[i]);
12    } else {
13      tube.enqueue({ uri: parentPath, delta: chars[i] }, isQuiet);
14    }
15    await sleep(50);
16  }
17 }
```

这样我们就了解了 Adapter 子模块的实现部分。

要点总结

这一节课，我们讲了 Ling 框架最底层的 Adapter 子模块，它实现了 OpenAI 和 Coze 两种 API 封装的兼容，在这两个封装实现当中，我们都是通过将 Tube 对象传给 getChatCompletions 函数，并创建 JSONParser 的方式处理流式数据的。

在下一节课，我们将继续介绍 Ling 框架的另外两个子模块，Bot 和 Tube。

课后练习

Ling 的底层为什么只实现了 OpenAI 和 Coze 两种封装呢？你是否用过或者看到过其他不一样的 API 规格呢？如果有的话，你可以将它分享出来，并尝试对其进行封装和兼容，并将你的收获分享到评论区。

AI智能总结

1. Ling框架包含四个子系统：adapter、bot、parser和tube，它们协同工作来处理复杂的AI工作流。
2. Adapter模块是底层的模块，负责调用大模型对话接口，兼容OpenAI和Coze两种规格，覆盖了常用的文本大模型。

3. Ling对象实例通过管理bot来处理节点输入输出，调用adapter选择具体的API调用，并通过parser动态解析大模型输入输出，最后通过tube发送数据给前端。
4. 代码实现了兼容OpenAI的API调用逻辑主体部分，包括处理消息、配置信息和输出格式，以及通过OpenAI SDK调用对话主体部分。
5. 重点关注Adapter模块的具体实现代码，因为它是执行大模型对话的底层模块。
6. 代码中的ChatConfig和ChatOptions对象定义了大模型的配置和对话输出内容的配置，包括模型名、服务地址、鉴权信息以及对输出内容的控制参数。
7. 代码中使用了条件判断来兼容两种子规格，分别是标准的OpenAI和微软的AzureOpenAI，通过判断config.endpoint来选择不同的对象创建方式。
8. 代码中使用了parser来解析内容，根据输出格式是否是JSON采用不同的解析方式，将解析内容存入缓冲区。
9. 代码中通过遍历events来解析内容进行处理，将内容放入缓冲对象buffer，然后再用一个异步过程，将对象中的内容一一放入Tube对象的流式队列里，实现内容的分步发送。
10. 代码中实现了对话结束的回调函数和parser解析内容中触发onStringResolved和onObjectResolved时调用相应的回调函数。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

精选留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。