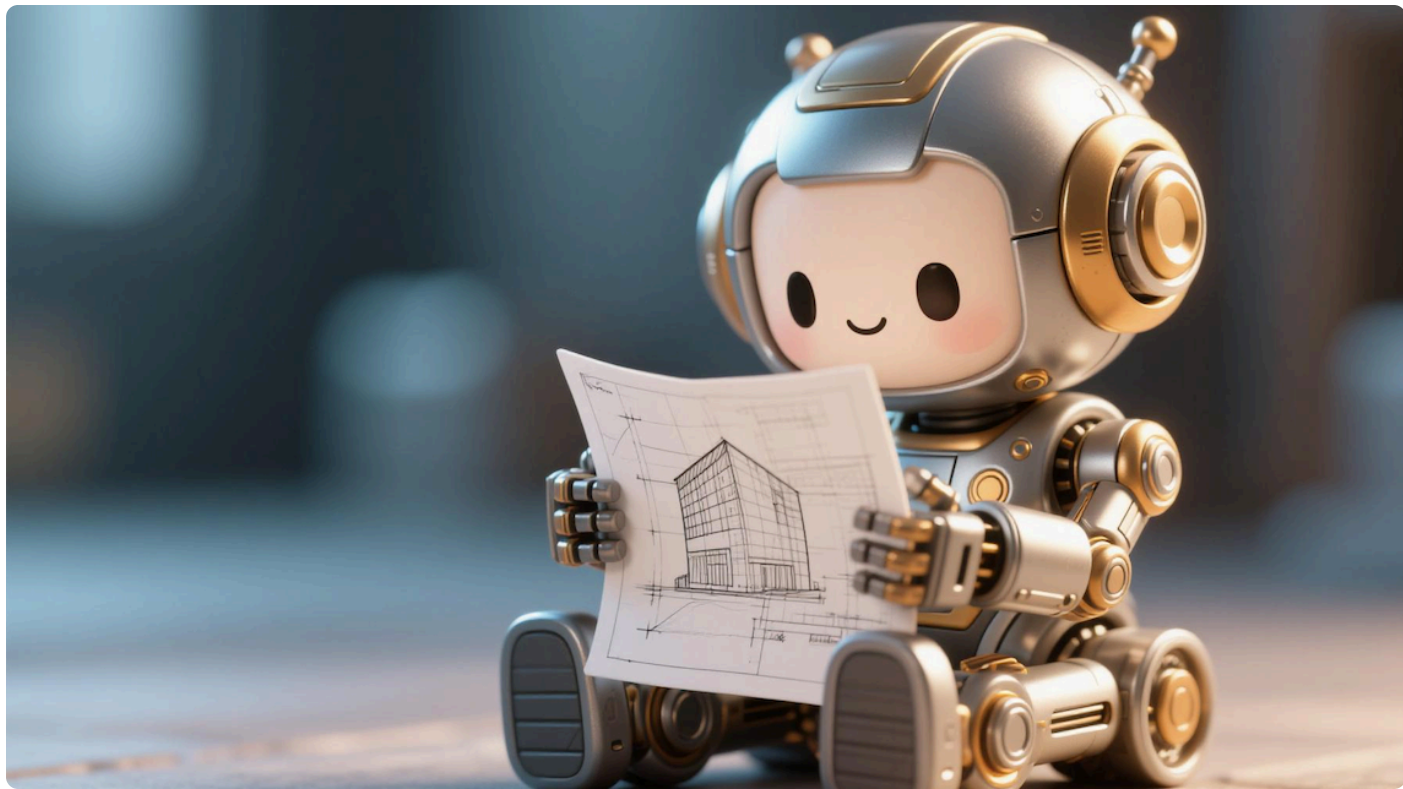


11 | workflow 框架：Ling 的设计思想与整体架构

月影 · 跟月影学前端智能体开发



你好，我是月影。

前面两节课我们讲了 AI 应用的用户体验中，非常核心的一个技术挑战，即如何在流式输出下，实时处理 JSON 格式的数据。

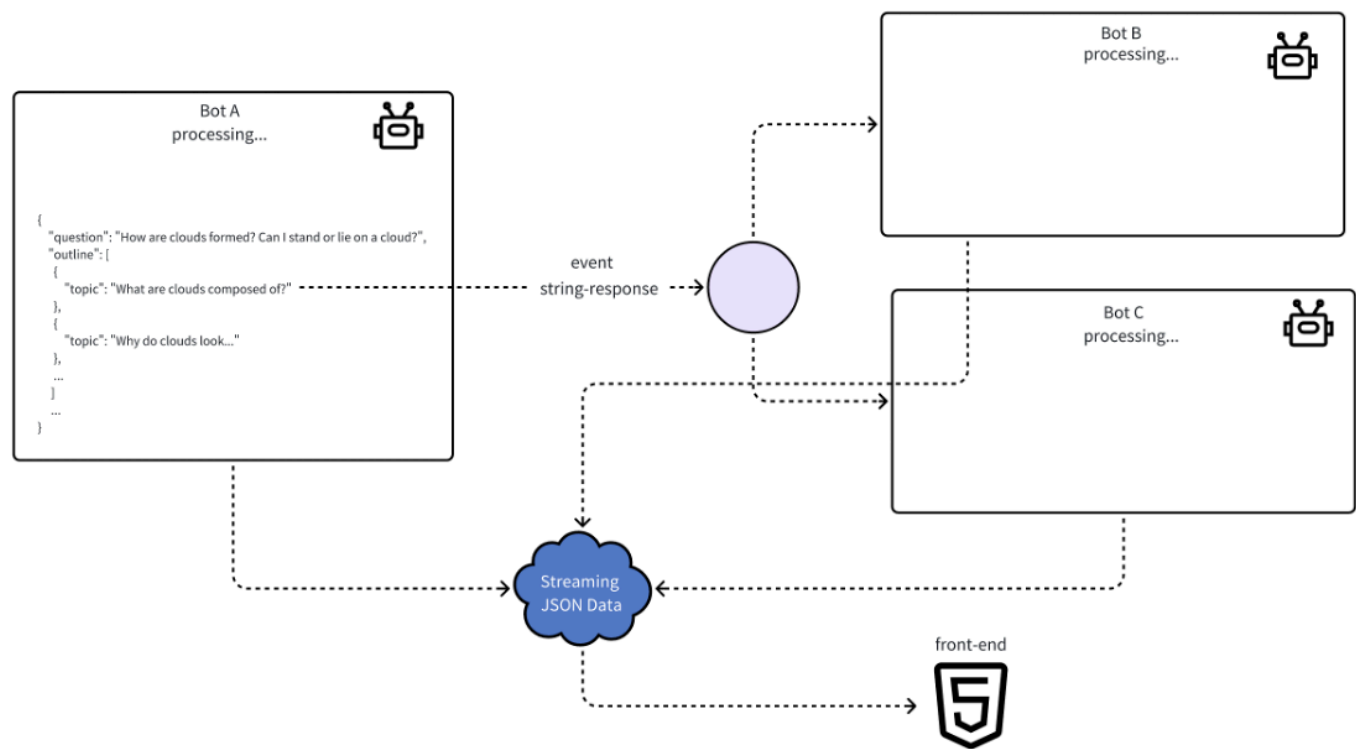
我采用的思路是自己实现 JSON 解析器，使用动态解析 JSON 数据流的方式，用 jsonuri 来动态构建 JSON 数据，这个方案构成处理整个 AI 业务工作流的核心。

基于这个核心，我封装了一个开源框架叫做 [🔗Ling](#)，用来方便地处理复杂的 AI 工作流。

由于在后续的 AI 应用实践课程中，我们会比较频繁地采用这个框架，因此在这一节课里，我先来介绍一下这个框架的设计思想与整体架构。

Ling 基础架构

Ling 是一个**基于流式 JSON 数据的异步工作流框架**。它能够比较方便地管理 AI 业务工作流的节点，并及时处理其中流转的数据。



如上图所示，业务中三个 AI 节点分别是 BotA、BotB、BotC，当 BotA 推理生成数据时，Ling 通过 string-response 事件在指定字段的数据输出完成时，及时分发给 BotB 和 BotC 进行处理。

在这个过程中，所有需要发送给客户端的数据，会通过单一的 Stream 实例完成数据的分发。

构建 Ling 工作流

这么说呢，还是比较抽象，我们还是看一个实际的例子。

让我们来实现一个简易的儿童版 AI 十万个为什么，用户输入一个问题，根据问题生成一篇适合儿童阅读的、内容稍微丰富一些的文章，效果如下：



我们看到，当我们输入一个问题“**天空为什么是蓝色的**”时，AI 能够非常快速地把内容实时生成出来。


实际上，这个是通过一个简单的工作流来实现的。我们用了两级 AI 节点，外层是一个大模型负责撰写大纲，内层是一组大模型，将大纲每一章节展开撰写。

具体如何做呢？你可以跟着我一起实际操作。

我们首先用 Trae 创建一个 Vue 项目。

然后安装依赖：

```
1 pnpm i @bearbobo/ling jsonuri
```

 复制代码

这样就安装了 Ling 框架以及客户端依赖的 jsonuri。


接着我们配置 `.env.local`：

```
1 VITE_API_KEY=sk-qi2oJBNF*****z8txbp4
2 VITE_END_POINT=https://api.moonshot.cn/v1/chat/completions
```

 复制代码

然后我们添加 `server.js`，内容如下：

```
1 import * as dotenv from 'dotenv'
2 import express from 'express';
3 import { Ling } from "@bearbobo/ling";
4 import { pipeline } from 'node:stream/promises';
5
6 dotenv.config({
7   path: ['.env.local', '.env']
8 })
9
10 const apiKey = process.env.VITE_API_KEY;
11 const endpoint = process.env.VITE_END_POINT;
12
13 const app = express();
14 const port = 3000;
15
16 const outlinePrompt = `
17 根据用户要求，生成科普文章大纲。
18
19 输出以下JSON格式内容：
```

 复制代码

```

20 {
21     "title": "文章标题",
22     "outline": [
23         {
24             "section": 1,
25             "title": "章节标题",
26             "subtopics": "子主题1\n子主题2\n子主题3",
27             "overview": "章节概述"
28         },
29         {
30             "section": 2,
31             "title": "章节标题",
32             "subtopics": "子主题1\n子主题2\n子主题3",
33             "overview": "章节概述"
34         },
35         {
36             "section": 3,
37             "title": "章节标题",
38             "subtopics": "子主题1\n子主题2\n子主题3",
39             "overview": "章节概述"
40         },
41         {
42             "section": 4,
43             "title": "总结",
44             "subtopics": "子主题1\n子主题2",
45             "overview": "章节概述"
46         },
47     ]
48 }
49 `;
50
51 const contentPrompt = `
52 根据用户发送的文章标题和概述，撰写详细文章内容。
53
54 要求：
55 文章的读者是6-8岁的儿童。
56 文章的风格要符合儿童的阅读习惯，避免使用过于复杂的句子结构和词汇。
57 文章的内容要围绕用户发送的文章标题和概述进行，不要偏离主题。
58 限制篇幅，不要超过3个自然段落，纯文本输出，不要加任何Markdown标签。
59 `;
60
61 // SSE 端点
62 app.get('/stream', async (req, res) => {
63     const question = req.query.question;
64
65     const config = {
66         model_name: 'moonshot-v1-8k',
67         api_key: apiKey,
68

```

```

69     endpoint: endpoint,
70     sse: true,
71 };
72
73 // ----- The work flow start -----
74 const ling = new Ling(config);
75
76 const outlineBot = ling.createBot();
77 outlineBot.addPrompt(outlinePrompt);
78
79 outlineBot.chat(question);
80
81 outlineBot.on('object-response', ({ uri, delta }) => {
82     const matches = uri.match(/outline\/(\d+)/);
83     if (matches) {
84         const section = matches[1];
85         console.log(uri, delta);
86         const contentBot = ling.createBot(`content/${section}`, {}, {
87             response_format: { type: "text" },
88         });
89         contentBot.addPrompt(contentPrompt);
90         contentBot.chat(`
91 # 主题
92 ${delta.title}
93
94 ## 子主题
95 ${delta.subtopics}
96
97 ## 摘要
98 ${delta.overview}
99     `);
100     }
101 });
102
103 ling.close();
104
105 // setting below headers for Streaming the data
106 res.writeHead(200, {
107     'Content-Type': "text/event-stream",
108     'Cache-Control': "no-cache",
109     'Connection': "keep-alive"
110 });
111
112 pipeline(ling.stream, res);
113 });
114
115 // 启动服务器
116 app.listen(port, () => {
117     console.log(`Server running on http://localhost:${port}`);

```

我们梳理一下代码逻辑。


首先刚才说有两级大模型，它们的系统提示词分别如下：

[📄 复制代码](#)

```
1  const outlinePrompt = `
2  根据用户要求，生成科普文章大纲。
3
4  输出以下JSON格式内容：
5
6  {
7      "title": "文章标题",
8      "outline": [
9          {
10             "section": 1,
11             "title": "章节标题",
12             "subtopics": "子主题1\n子主题2\n子主题3",
13             "overview": "章节概述"
14         },
15         {
16             "section": 2,
17             "title": "章节标题",
18             "subtopics": "子主题1\n子主题2\n子主题3",
19             "overview": "章节概述"
20         },
21         {
22             "section": 3,
23             "title": "章节标题",
24             "subtopics": "子主题1\n子主题2\n子主题3",
25             "overview": "章节概述"
26         },
27         {
28             "section": 4,
29             "title": "总结",
30             "subtopics": "子主题1\n子主题2",
31             "overview": "章节概述"
32         },
33     ]
34 }
35 `;
36
37 const contentPrompt = `
38 根据用户发送的文章标题和概述，撰写详细文章内容。
```

```
39
40 要求：
41 文章的读者是6-8岁的儿童。
42 文章的风格要符合儿童的阅读习惯，避免使用过于复杂的句子结构和词汇。
43 文章的内容要围绕用户发送的文章标题和概述进行，不要偏离主题。
44 限制篇幅，不要超过3个自然段落，纯文本输出，不要加任何Markdown标题。
45 `;
```

接着呢，我们用配置创建一个 Ling 实例，然后创建一个 outlineBot，负责大纲的撰写。

 复制代码

```
1  const question = req.query.question;
2
3  const config = {
4      model_name: 'moonshot-v1-8k',
5      api_key: apiKey,
6      endpoint: endpoint,
7      sse: true,
8  };
9
10 // ----- The work flow start -----
11 const ling = new Ling(config);
12
13 const outlineBot = ling.createBot();
14 outlineBot.addPrompt(outlinePrompt);
15
16 outlineBot.chat(question);
```

使用 Ling 创建工作流非常简单，我们直接用配置创建 Ling 的实例对象，然后调用该对象的 createBot 方法，就可以创建出一个由 Ling 托管的大模型节点对象。


注意这里的 Bot 对象，可以用传给 createBot 独立的 api_key、endpoint 和 model_name 参数的方式创建出不同的大模型对象，但是不传的话，则默认使用从 Ling 实例继承的配置，这里我们配置的是 Kimi 大模型。

接着我们通过 addPrompt 方法将 outlinePrompt 传入，这个方法默认支持 nunjucks 模板，所以它的第二个参数可以传一个对象，不过我们这个例子没有用到动态参数。

接着我们调用 chat 方法，传入文本就可以了。

注意 Ling 默认做了 OpenAI 和 Coze 接口的兼容，所以我们创建 Ling 时，model_name、apk_key 和 endpoint 可以传 DeepSeek、Kimi (moonshot) 或豆包，以及其他任何兼容 OpenAI 的大模型。当我们使用豆包时，model_name 要传 botId。

接下来，我们需要监听 outlineBot 的 `object-response` 事件：

 复制代码

```
1 outlineBot.on('object-response', ({ uri, delta }) => {
2     const matches = uri.match(/outline\/(\d+)/);
3     if (matches) {
4         const section = matches[1];
5         const contentBot = ling.createBot(`content/${section}`, {}, {
6             response_format: { type: "text" },
7         });
8         contentBot.addPrompt(contentPrompt);
9         contentBot.chat(`
10 # 主题
11 ${delta.title}
12
13 ## 子主题
14 ${delta.subtopics}
15
16 ## 摘要
17 ${delta.overview}
18     `);
19     }
20 });
```

该事件在 JSON 解析完成某个数组或对象属性时被自动触发。

除了 `object-response` 外，Ling 还支持 `string-response` 和 `inference-done` 事件，前者在解析完成某个字符串属性时被触发，相当于前面课程 JSONParser 的 `string-resolve` 事件；后者在整个 Bot 推理完成时触发。

因为我们在文章每个小节完成提纲生成时，就可以立即发给生成正文的 Bot 处理。所以这里使用了 `object-response` 事件，确保它第一时间就被后续节点立即处理，这样减少等


待时间。

在 `object-response` 事件中，我们根据事件 `uri` 参数判断是否是章节大纲，若是，那么 `uri` 对应的应该是 `/outline/1`、`/outline/2` 这样的路径，我们可以通过正则匹配出来。

然后我们创建二级 Bot 用来处理正文。因为这里的正文输出不需要 JSON 格式，所以我们通过 `response_format: { type: "text" }` 强制 Bot 输出文本。我们通过设置 `createBot` 的第一个参数 `root` 为 `content/${section}` 将文本输出到 `/content` 数组中的 `section` 下标对应元素里。

接着我们调用 `ling.close()` 将流结束，这个操作会发送 `finished` 事件给客户端，客户端就可以结束 `SourceEvent`。

最后我们将 `ling.stream` 通过 `pipeline` 进行流式连接，并设置好对应的 HTTP Header，这样就可以正常发送数据流给客户端了。

 复制代码

```
1 ling.close();
2
3 // setting below headers for Streaming the data
4 res.writeHead(200, {
5     'Content-Type': "text/event-stream",
6     'Cache-Control': "no-cache",
7     'Connection': "keep-alive"
8 });
9
10 pipeline(ling.stream, res);
```

实现前端 UI 交互

我们在创建 `Ling` 实例的时候配置了 `sse` 参数，所以接口返回的数据格式是 `Server-Sent Events` 格式，在前端使用起来会比较方便。

我们配置 `vite.config.js` 转发 `server` 接口：

```
1  server: {
2    allowedHosts: true,
3    port: 4399,
4    proxy: {
5      '/api': {
6        target: 'http://localhost:3000',
7        secure: false,
8        rewrite: path => path.replace(/^\/api/, ''),
9      },
10   },
11  },
```

然后改写 App.vue。

```
1  <script setup lang="ts">
2  import { ref } from 'vue';
3  import { set, get } from 'jsonuri';
4
5  const question = ref('天空为什么是蓝色的? ');
6  const content = ref({
7    title: "",
8    outline: [],
9    content: [],
10  });
11
12  const update = async () => {
13    if (!question) return;
14
15    const endpoint = '/api/stream';
16
17    const eventSource = new EventSource(`${endpoint}?question=${question.value}`);
18    eventSource.addEventListener("message", function (e: any) {
19      let { uri, delta } = JSON.parse(e.data);
20      const str = get(content.value, uri);
21      set(content.value, uri, (str || '') + delta);
22    });
23    eventSource.addEventListener('finished', () => {
24      console.log('传输完成');
25      eventSource.close();
26    });
27  }
28  </script>
```

```
29 <template>
30   <div class="container">
31     <div>
32       <label>输入: </label><input class="input" v-model="question" />
33       <button @click="update">提交</button>
34     </div>
35     <div class="output">
36       <!-- <textarea>{{ content }}</textarea> -->
37       <h1>{{ content.title }}</h1>
38       <div v-for="(item, i) in (content.outline as any)" :key="item.title + i">
39         <h2>{{ item.title }}</h2>
40         <p>{{ content.content[i] }}</p>
41       </div>
42     </div>
43   </div>
44 </template>
45
46 <style scoped>
47 .container {
48   display: flex;
49   flex-direction: column;
50   align-items: start;
51   justify-content: start;
52   height: 100vh;
53   font-size: .85rem;
54 }
55
56 .input {
57   width: 200px;
58 }
59
60 .output {
61   margin-top: 10px;
62   min-height: 300px;
63   width: 100%;
64   text-align: left;
65 }
66
67 button {
68   padding: 0 10px;
69   margin-left: 6px;
70 }
71
72 textarea {
73   width: 300px;
74   height: 200px;
75   font-size: 10px;
76 }
77
```

客户端代码非常简单，核心还是 EventSource 处理：

[复制代码](#)

```
1  const eventSource = new EventSource(`${endpoint}?question=${question.value}`);
2  eventSource.addEventListener("message", function (e: any) {
3      let { uri, delta } = JSON.parse(e.data);
4      const str = get(content.value, uri);
5      set(content.value, uri, (str || '') + delta);
6  });
7  eventSource.addEventListener('finished', () => {
8      console.log('传输完成');
9      eventSource.close();
10 });
```

注意一个小细节，我们之前自己写的 SSE 实现中，结束事件我们用的是 `end`，而 Ling 中默认用 `finished`，所以我们要把监听 `end` 事件改成监听 `finished` 事件。


这样我们就实现了简易儿童版 AI 十万个为什么。

前端渲染 Markdown 内容

再补充一个细节，我们在生成章节内容的大模型节点中禁止模型输出任何 Markdown 标签，因为我们不想让这个模型输出标题，否则它就和大纲的章节标题重复了。但是如果只是禁止输出标题，允许正文中有一些 Markdown 标签（比如字体加粗），那么我们怎么在前端展示呢？


我们可以将 Markdown 通过 `marked` 库转成 HTML 然后进行渲染，这也是 AI 应用常用的前端内容渲染方式。

我们安装 `marked` 库：

 复制代码

```
1 pnpm i marked
```

然后改写 App.vue，添加引用并改写内容渲染方式。

 复制代码

```
1 <script setup lang="ts">
2 import { marked } from 'marked';
3 ...
4 </script>
5
6 <template>
7 ...
8     <div v-for="(item, i) in (content.outline as any)" :key="item.title + i">
9         <h2>{{ item.title }}</h2>
10        <p v-html=marked(content.content[i])></p>
11    </div>
12 ...
13 </template>
```

这样我们就可以在前端展示大模型输出的 Markdown 内容了。

要点总结

这节课我们学习了 Ling 框架的基础架构，通过完整的例子，了解了如何使用 Ling 来构建工作流并实现前端交互。我们可以看到，Ling 是一个非常方便易用的异步实时响应工作流框架。在后续的课程中我们还会继续使用它。

Ling 的完整代码在 GitHub 仓库： [🔗https://github.com/WeHomeBot/ling](https://github.com/WeHomeBot/ling)

有兴趣的同学可以进一步深入研究它。我希望你不仅能够掌握 Ling 的用法，还能了解 Ling 的设计细节，有能力的同学欢迎一起改进 Ling 的代码，让它变得更加方便和强大。

课后练习

Ling 的 createBot 默认只支持创建文本大模型对象，假设我们需要处理图片，给上面的例子增加配图功能，为每一篇文章生成一张封面图，我们应该怎么做呢？你可以课后动手修改项目代码试试看，遇到任何问题请分享到评论区，我会帮你解答。

完整的项目代码在 https://github.com/akira-cn/frontend-dev-large-model-era/tree/main/ling_sse

AI智能总结

1. Ling框架是基于流式JSON数据的异步工作流框架，旨在方便地管理AI业务工作流的节点，并及时处理其中流转的数据。
2. Ling框架的设计思想是基于流式JSON数据的异步处理，通过创建Bot节点来处理不同的任务，实现工作流的管理和数据处理。
3. Ling框架的应用举例包括创建一个儿童版AI十万个为什么的工作流，通过两级AI节点实现用户输入问题后快速生成适合儿童阅读的文章。
4. Ling框架的使用方法涵盖配置Ling实例对象、创建Bot节点、添加任务提示、触发任务处理、监听事件等简单操作。
5. Ling框架支持多种事件类型，如`object-response`、`string-response`和`inference-done`，用于处理JSON解析完成、字符串解析完成和整个Bot推理完成等事件。
6. Ling框架的流式数据传输通过设置HTTP Header和使用pipeline进行流式连接，实现数据流的发送和处理。
7. Ling框架的前端交互实现使用了EventSource处理，通过配置sse参数，接口返回的数据格式是Server-Sent Events格式，便于前端使用。
8. Ling框架的前端渲染Markdown内容通过使用marked库将Markdown转成HTML进行渲染，实现了在前端展示大模型输出的Markdown内容。
9. Ling框架是一个非常方便易用的异步实时响应工作流框架，具有广泛的应用前景和可扩展性。
10. Ling框架的完整代码可在GitHub仓库获取，有兴趣的同学可以进一步深入研究，掌握其用法和设计细节。

© 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。