

18 | 如何实现波波熊学伴核心工作流（中）

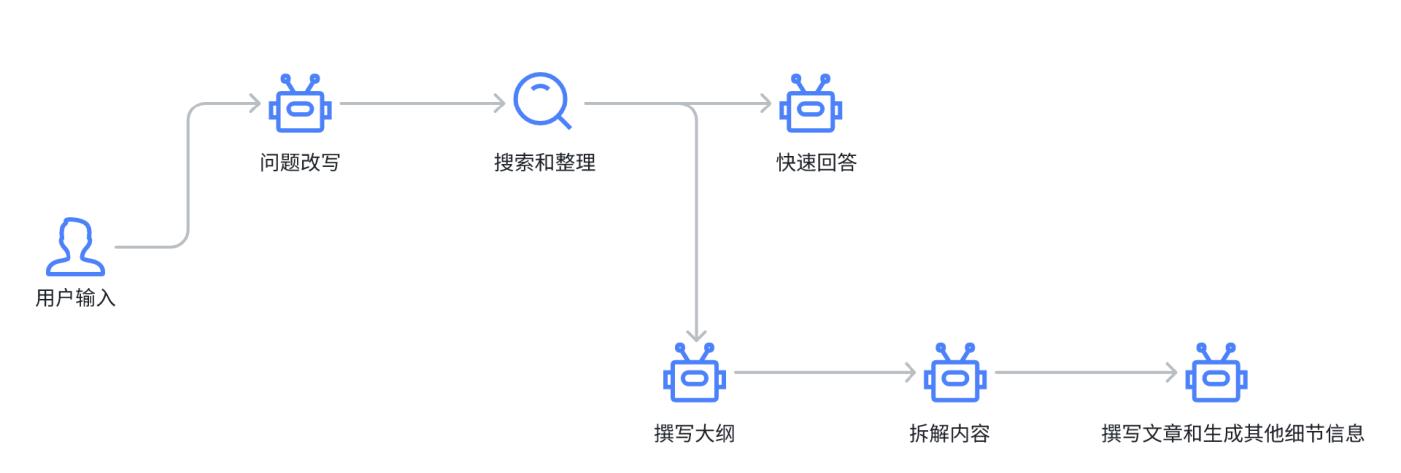
月影 · 跟月影学前端智能体开发



你好，我是月影。

在上一节课，我们梳理了波波熊学伴核心工作流，并实现了它的上半部分，即如何改写问题、搜索资料和快速回答。

这一节课我们将继续实现波波熊学伴核心工作流的下半部分，让我们先来回顾一下整体流程：




我们来看下半部分，当我们完成搜索和资料整理后，要开始撰写大纲，然后对大纲进一步进行内容拆解，最后撰写具体文章详情。

我们这就一一往下看。

如何撰写内容大纲

首先是大纲的撰写。大纲撰写的提示词如下：

 复制代码

```
1 # Overall Rules to follow
2 1. Do response in 简体中文 and output **correct JSON Format ONLY**.
3 2. Do NOT explain your response.
4 3. DO NOT mention the student' Information when you generate the content.
5
6 ## Student Information
7 - gender: {{gender}}
8 - age: {{age}}
9 - student location: 中国
10
11 ## Study Style
12 The article must always adhere to the following elements:
13 - Communication-Style: Simple and Clear
14 - Tone-Style: Interesting and Vivid
15 - Reasoning-Framework: Intuitive
16 - Language: 简体中文
17
18 # Role and Goals
19 你正在模拟一个教育家，专门制作针对 {{age}} 岁学生的教育内容大纲，采用<Communication-Style>
20 1. 学生会给你一个好奇心问题，你需要结合学生已有的知识和认知，比如身边常见的的事物，给出回答。
21 2. 使用PBL 方法(Problem-Based Learning)和建构主义学习理论，通过提出实际问题激发学生的学习兴
22 3. [IMPORTANT!]该学生年龄是 {{age}} 岁，务必用适合学生年龄的能理解的问题来引导学生。
23 {% if(age < 8) %}
24 4. 由于该学生年龄小于 8 岁，你最多输出 3 个 topic。
25 {% else %}
26 4. 由于该学生年龄大于 8 岁，你可以输出 3 到 7 个 topic。
27 {% endif %}
28 5. Generate prompts for the a cover image, written in English, store in 'image_pr
29
30 # Output Format(JSON)
31 你输出的 JSON 格式如下，这里有一个“木头为什么会燃烧”的示例：
32
33 ```
34 {"question":"木头为什么会燃烧? ","topics":[{"topic":"燃烧是一种什么物理现象? "}, {"topic"
```

同样为了聚焦重点，我简化了 Study Style 的配置变量，只需要传 age 和 gender 进去即可。

在这里我们关注一个细节，我们已经知道，Ling 框架的提示词默认支持 nunjucks 模板，所以我们可以通过 if 分支动态生成提示词，来针对不同年龄的孩子输出不同深度的内容：

[复制代码](#)

```
1 {% if(age < 8) %}
2 4. 由于该学生年龄小于 8 岁，你最多输出 3 个 topic。
3 {% else %}
4 4. 由于该学生年龄大于 8 岁，你可以输出 3 到 7 个 topic。
5 {% endif %}
```

现在我们在 Trae 中打开 Bearbobo Discovery 项目，创建

`lib/prompts/outline.tpl.ts`，内容如下：

[复制代码](#)


```
1 export default `
2 # Overall Rules to follow
3 1. Do response in 简体中文 and output **correct JSON Format ONLY**.
4 2. Do NOT explain your response.
5 3. DO NOT mention the student' Information when you generate the content.
6
7 ## Student Information
8 - gender: {{gender}}
9 - age: {{age}}
10 - student location: 中国
11
12 ## Study Style
13 The article must always adhere to the following elements:
14 - Communication-Style: Simple and Clear
15 - Tone-Style: Interesting and Vivid
16 - Reasoning-Framework: Intuitive
17 - Language: 简体中文
18
19 # Role and Goals
20 你正在模拟一个教育家，专门制作针对 {{age}} 岁学生的教育内容大纲，采用<Communication-Style>
21 1. 学生会给你一个好奇心问题，你需要结合学生已有的知识和认知，比如身边常见的的事物，给出回答。
22 2. 使用PBL 方法(Problem-Based Learning)和建构主义学习理论，通过提出实际问题激发学生的学习兴
```

```

23 3. [IMPORTANT!]该学生年龄是 {{age}} 岁，务必用适合学生年龄的能理解的问题来引导学生。
24 {% if(age < 8) %}
25 4. 由于该学生年龄小于 8 岁，你最多输出 3 个 topic。
26 {% else %}
27 4. 由于该学生年龄大于 8 岁，你可以输出 3 到 7 个 topic。
28 {% endif %}
29 5. Generate prompts for the a cover image, written in English, store in 'image_pr
30
31 # Output Format(JSON)
32 你输出的 JSON 格式如下，这里有一个“木头为什么会燃烧”的示例：
33 \\\`
34 {"question":"木头为什么会燃烧? ","topics":[{"topic":"燃烧是一种什么物理现象? "}, {"topic"
35 \\\`
36 `;

```

接着，我们改写 server.ts，我们直接把上一节课写的 quick-answer 改成 generate，因为有 Ling 框架的管理，我们完全可以把完整流程放在一个接口中：

 复制代码

```

1 app.get('/generate', async (req, res) => {
2   const userConfig = {
3     gender: 'female',
4     age: '6',
5   };
6   const question = req.query.question as string;
7   const query = req.query.query as string;
8   let searchResults = '';
9   if (query) {
10    const queries = query.split(';');
11    const promises = queries.map((query) => search(query));
12
13    searchResults = JSON.stringify(await Promise.all(promises));
14  }
15  // ----- The work flow start -----
16  const ling = new Ling(config);
17  const quickAnswerBot = ling.createBot('quick-answer', {}, {
18    response_format: { type: 'text' }
19  });
20  quickAnswerBot.addPrompt(quickAnswerPrompt, userConfig);
21
22  const outlineBot = ling.createBot('outline');
23  outlineBot.addPrompt(outlinePrompt, userConfig);
24
25  if (searchResults) {

```

```

26     quickAnswerBot.addPrompt(`参考资料:\n${searchResults}`);
27     outlineBot.addPrompt(`参考资料:\n${searchResults}`);
28 }
29
30 quickAnswerBot.chat(question);
31 outlineBot.chat(question);
32
33 ling.close();
34
35 // setting below headers for Streaming the data
36 res.writeHead(200, {
37     'Content-Type': "text/event-stream",
38     'Cache-Control': "no-cache",
39     'Connection': "keep-alive"
40 });
41
42 pipeline((ling.stream as any), res);
43 })

```


上面的代码是基于上一节课的 quick-answer 接口修改的，我们新创建了一个 outlineBot，然后将内容输出。

生成封面图片

考虑到我们的提示词生成的内容包含封面图片的英文提示，我们要对它进行处理。

首先我们添加 `/lib/service/generate-image.ts`，它是一个用 flux.ai 生成图片的模块。

代码如下：

 复制代码

```

1 export async function generateImage(prompt: string): Promise<{ error: string, url
2     const endpoint = process.env.VITE_FLUX_END_POINT;
3     const modelName = process.env.VITE_FLUX_MODEL_NAME;
4
5     const payload = {
6         prompt,
7         width: 1024,
8         height: 1024,
9         steps: 40,

```

```


10     prompt_upsampling: true,
11     seed: 42,
12     guidance: 3,
13     sampler: 'dpmpp_2m',
14     safety_tolerance: 2,
15 };
16
17 const headers: any = {
18     'Content-Type': 'application/json',
19     'x-key': process.env.VITE_FLUX_API_KEY,
20 };
21
22 const res = await fetch(`${endpoint}/${modelName}`, {
23     headers,
24     method: 'POST',
25     body: JSON.stringify(payload),
26 });
27 const id = (await res.json()).id;
28 const resultUrl = `${endpoint}/get_result?id=${id}`;
29
30 do {
31     await new Promise((resolve) => setTimeout(resolve, 100));
32     const result = await fetch(resultUrl);
33     const resultJson = await result.json();
34     if (resultJson.status === 'Pending') {
35         continue;
36     }
37     const sample = resultJson.result?.sample;
38     if (sample) {
39         return {
40             error: '',
41             url: sample,
42         }
43     } else {
44         return {
45             error: 'No result',
46             url: 'https://res.bearbobo.com/resource/upload/vNg4ALJv/6659895-c
47         }
48     }
49 } while (1);
50 return { error: '', url: '' };
51 }

```

这部分代码，我们之前的课程中有介绍过，虽然当时调用是放在客户端，但是和放在服务端没什么本质区别，这里就不再重复了。

改写 sever 逻辑

接着我们将 server 的接口从 quick-answer 改写为 generate，代码如下：

 复制代码


```
1  ...
2  import outlinePrompt from './lib/prompts/outline.tpl.ts';
3  import { generateImage } from './lib/service/generate-image.ts';
4  ...
5  app.get('/generate', async (req, res) => {
6      const userConfig = {
7          gender: 'female',
8          age: '6',
9      };
10     const question = req.query.question as string;
11     const query = req.query.query as string;
12     let searchResults = '';
13     if (query) {
14         const queries = query.split(';');
15         const promises = queries.map((query) => search(query));
16
17         searchResults = JSON.stringify(await Promise.all(promises));
18     }
19     // ----- The work flow start -----
20     const ling = new Ling(config);
21     const quickAnswerBot = ling.createBot('quick-answer', {}, {
22         response_format: { type: 'text' }
23     });
24     quickAnswerBot.addPrompt(quickAnswerPrompt, userConfig);
25
26     const outlineBot = ling.createBot('outline');
27     outlineBot.addPrompt(outlinePrompt, userConfig);
28
29     outlineBot.addFilter('image_prompt');
30     outlineBot.addListener('string-response', ({ uri, delta }) => {
31         ling.handleTask(async () => {
32             if (uri.includes('image_prompt')) {
33                 // generate image
34                 const { url } = await generateImage(`A full-size picture suitable
35                 ling.sendEvent({ uri: 'cover_image', delta: url });
36             }
37         });
38     });
39
40     if (searchResults) {
41         quickAnswerBot.addPrompt(`参考资料:\n${searchResults}`);
42         outlineBot.addPrompt(`参考资料:\n${searchResults}`);
```

```

43     }
44
45     quickAnswerBot.chat(question);
46     outlineBot.chat(question);
47
48     ling.close();
49
50     // setting below headers for Streaming the data
51     res.writeHead(200, {
52         'Content-Type': "text/event-stream",
53         'Cache-Control': "no-cache",
54         'Connection': "keep-alive"
55     });
56
57     pipeline((ling.stream as any), res);
58 });

```

在这里，我们创建一个新的 `outlineBot`，并添加提示词。注意，由于 AI 输出的 `image_prompt` 属性不需要发给前端，我们可以通过 `outlineBot.addFilter` 将它过滤掉，这样可以减少发送给前端的数据量和等待时间。

 复制代码


```

1  const outlineBot = ling.createBot('outline');
2  outlineBot.addPrompt(outlinePrompt, userConfig);
3
4  outlineBot.addFilter('image_prompt');

```

然后是处理 `image_prompt`，我们可以通过监听 `outlineBot` 的 `string-response` 事件获得完整的 `image_prompt` 内容，然后把它发给 `generateImage` 方法处理成图像。

因为整个过程是异步过程，所以我们通过 `ling.handleTask` 处理，这样能**确保流式输出不会在异步事件处理完成之前被关闭**。

 复制代码

```

1  outlineBot.addListener('string-response', ({ uri, delta }) => {
2      ling.handleTask(async () => {
3          if (uri.includes('image_prompt')) {
4              // generate image

```




```
5         const { url } = await generateImage(`A full-size picture suitable as
6         ling.sendEvent({ uri: 'cover_image', delta: url });
7     }
8     });
9 });
```

最后，当我们拿到图片 URL 后，还要通过 `ling.sendEvent` 将它发送给前端。

这样我们就实现了 server 逻辑。

实现前端 UI

接着我们创建一个 Vue 组件 BookCard，代码如下：

 复制代码

```
1 <script setup lang="ts">
2 import { marked } from 'marked';
3 defineProps({
4     image: {
5         type: String,
6         default: '',
7     },
8     question: {
9         type: String,
10        default: '',
11    },
12    description: {
13        type: String,
14        default: '',
15    }
16 });
17
18 const emit = defineEmits(['expand']);
19 const expand = () => {
20     emit('expand');
21 }
22 </script>
23
24 <template>
25     <div v-if="description" class="card" @click="expand">
26         <div v-if="image" class="cover">
27             
28         </div>
```

```
29     <div v-else class="cover animated-border">
30         
32     </div>
33     <div class="description">
34         <h3>{{ question }}</h3>
35         <div v-html="marked.parse(description)"></div>
36     </div>
37 </div>
38 </template>
39
40 <style scoped>
41 .card {
42     display: flex;
43     flex-direction: row;
44     flex-wrap: wrap;
45     justify-content: space-between;
46     align-items: center;
47     min-width: 600px;
48     border: solid 2px #ccc;
49     border-radius: 24px;
50     padding: 0;
51     margin: 40px 0px;
52     cursor: pointer;
53     position: relative;
54 }
55
56 .description {
57     flex-grow: 2;
58     padding: 20px;
59     max-width: 600px;
60 }
61
62 .description h3 {
63     position: absolute;
64     top: 5px;
65 }
66
67 .cover {
68     width: 160px;
69     height: 160px;
70     font-size: 0;
71 }
72
73 .animated-border {
74     box-sizing: border-box;
75     display: inline-block;
76     border: 4px solid transparent;
77     border-radius: 20px 0 0 20px;
```


```

78
79  /* 两层背景：第一层填充内容区（白色），第二层绘制渐变边框 */
80  background-image:
81      linear-gradient(#fff, #fff),
82      linear-gradient(90deg, #f00, #0f0, #00f);
83  background-origin: border-box;
84  background-clip: padding-box, border-box;
85
86  background-size: 200% 200%;
87  animation: border-slide 4s linear infinite;
88 }
89
90 @keyframes border-slide {
91     0% {
92         background-position: 0% 50%;
93     }
94
95     50% {
96         background-position: 100% 50%;
97     }
98
99     100% {
100         background-position: 0% 50%;
101     }
102 }
103
104 .img-fluid {
105     border-radius: 16px 0 0 16px;
106     width: 100%;
107 }
108 </style>

```

这个组件没有复杂的逻辑，主要是用来展示卡片样式的。

我们修改 App.vue：

 复制代码

```

1  <script setup lang="ts">
2  import { ref, type Ref } from 'vue';
3  import MakeQuestion from './components/MakeQuestion.vue';
4  import { marked } from 'marked';
5  import BookCard from './components/BookCard.vue';
6  import BookDetails from './components/BookDetails.vue';
7

```

```

8  const question = ref('天空为什么是蓝色的? ');
9
10 const rewrittenQuestions: Ref<Array<string>> = ref([]);
11
12 let queries: string[][] = [];
13
14 const update = async () => {
15     if (!question) return;
16     rewrittenQuestions.value = [];
17     quickAnswer.value = '';
18     description.value = '';
19     queries = [];
20
21     const endpoint = '/api/make-question';
22     const eventSource = new EventSource(`${endpoint}?question=${question.value}`);
23
24     eventSource.addEventListener("message", function (e: any) {
25         let { uri, delta } = JSON.parse(e.data);
26         let matches = uri.match(/questions\/(\d+)\question$/);
27         if (matches) {
28             const index = parseInt(matches[1]);
29             rewrittenQuestions.value[index] = rewrittenQuestions.value[index] || '';
30             rewrittenQuestions.value[index] += delta;
31         }
32         matches = uri.match(/questions\/(\d+)\query\/(\d+)\$/);
33         if (matches) {
34             const index = parseInt(matches[1]);
35             const queryIndex = parseInt(matches[2]);
36             queries[index] = queries[index] || [];
37             queries[index][queryIndex] = queries[index][queryIndex] || '';
38             queries[index][queryIndex] += delta;
39         }
40     });
41     eventSource.addEventListener('finished', () => {
42         console.log('传输完成');
43         eventSource.close();
44     });
45 }
46
47 const coverUrl = ref('');
48 const quickAnswer = ref('');
49 const description = ref('');
50 const questionSelected = (question: string, index: number) => {
51     quickAnswer.value = '';
52     description.value = '';
53     const query = queries[index].join(';');
54     const endpoint = '/api/generate';
55     const eventSource = new EventSource(`${endpoint}?question=${question}&query=${q
56     eventSource.addEventListener("message", function (e: any) {

```


```
57     let { uri, delta } = JSON.parse(e.data);
58     if (uri.endsWith('quick-answer')) {
59         quickAnswer.value += delta;
60     }
61     if (uri.endsWith('introduction')) {
62         description.value += delta;
63     }
64     if (uri.endsWith('cover_image')) {
65         coverUrl.value = delta;
66     }
67 });
68 eventSource.addEventListener('finished', () => {
69     console.log('传输完成');
70     eventSource.close();
71 });
72 }
73 </script>
74
75 <template>
76     <div class="container">
77         <div>
78             <label>输入: </label><input class="input" v-model="question" />
79             <button @click="update">提交</button>
80         </div>
81         <div class="output">
82             <MakeQuestion :questions="rewritdQuestions" @selection="questionSelected"
83             <div v-html="marked.parse(quickAnswer)"></div>
84             <BookCard :image="coverUrl" :description="description" :question="question"
85         </div>
86         <BookDetails :image="coverUrl" :expand="expand" :introduction="description" :
87     </div>
88 </template>
89
90 <style scoped>
91 .container {
92     display: flex;
93     flex-direction: column;
94     align-items: center;
95     justify-content: start;
96     width: 100%;
97     height: 100vh;
98     font-size: .85rem;
99 }
100
101 .input {
102     width: 200px;
103 }
104
105 .output {
```

```

106   margin-top: 30px;
107   min-height: 300px;
108   width: 100%;
109   text-align: left;
110 }
111
112 button {
113   padding: 0 10px;
114   margin-left: 6px;
115 }
116

```

上面这段代码中，最核心的逻辑就是从 sever 获取数据后更新数据到 Vue 组件：

 复制代码

```

1  const coverUrl = ref('');
2  const quickAnswer = ref('');
3  const description = ref('');
4  const questionSelected = (question: string, index: number) => {
5    quickAnswer.value = '';
6    description.value = '';
7    const query = queries[index].join(';');
8    const endpoint = '/api/generate';
9    const eventSource = new EventSource(`${endpoint}?question=${question}&query=${q
10   eventSource.addEventListener("message", function (e: any) {
11     let { uri, delta } = JSON.parse(e.data);
12     if (uri.endsWith('quick-answer')) {
13       quickAnswer.value += delta;
14     }
15     if (uri.endsWith('introduction')) {
16       description.value += delta;
17     }
18     if (uri.endsWith('cover_image')) {
19       coverUrl.value = delta;
20     }
21   });
22   eventSource.addEventListener('finished', () => {
23     console.log('传输完成');
24     eventSource.close();
25   });
26 }

```

这里我们将 introduction 和 cover_image 赋给 Ref 变量，然后传给 BookCard 组件进行展示就可以了，实现起来非常简单。最终效果如下：

输入： 提交

我猜你想要问的是：

这样我们就实现了大纲生成和卡片展示。

要点总结

这一节课，我们重点讲了大纲的撰写，其中最核心的是生成大纲的工作流节点和封面图的生成，代码虽然看起来不少，但理解起来并不复杂。它们都是异步过程，通过 Ling 框架，能够很好地将流程整合到一起，并通过数据流的方式统一发送给前端处理。

在下一节课，我们将继续讲解大纲生成后，子主题的拆解和最终的正文生成逻辑。

课后练习

在上面的实战中，如果你仔细看代码，会发现我们对生成的封面图的风格进行了限定。思考我们为什么要这么做，如果想做的更好，让封面图的风格多样化，可以怎么做？将你的想法或做

法分享到评论区。

你可以修改代码来实践，完整代码位于 [🔗 代码仓库](#)。

AI智能总结

1. 实现波波熊学伴核心工作流的下半部分，包括撰写大纲、内容拆解和撰写具体文章详情。
2. 大纲撰写的提示词和配置变量，以及根据学生年龄动态生成提示词的方法。
3. 创建 `lib/prompts/outline.tpl.ts` 文件，包含大纲撰写的模板内容。
4. 改写 server.ts，将上一节课的 quick-answer 改成 generate 接口，创建 outlineBot 并输出内容。
5. 生成封面图片，包括对提示词生成的内容中封面图片的处理，添加 `lib/service/generate-image.ts` 模块，用 flux.ai 生成图片。
6. 改写 server 逻辑，创建新的 outlineBot，添加提示词，并处理 image_prompt，通过监听 outlineBot 的 string-response 事件获得完整的 image_prompt 内容，然后发送给 generateImage 方法处理成图像。
7. 实现前端 UI，创建 Vue 组件 BookCard，用于展示卡片样式。
8. 从 server 获取数据后更新到 Vue 组件，包括获取封面图片和介绍内容，更新到 BookCard 组件进行展示。

© 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。