

17 | 如何实现波波熊学伴核心 workflow（上）

月影 · 跟月影学前端智能体开发



你好，我是月影。

上一节课，我们介绍了如何通过问题改写和 RAG 来让大模型减少幻觉，给出质量高的内容。那么这节课，我们要继续深入，完成波波熊学伴产品整体的核心 workflow。

现在我们通过 search 得到了参考资料，我们让大模型按照两条线并行处理。

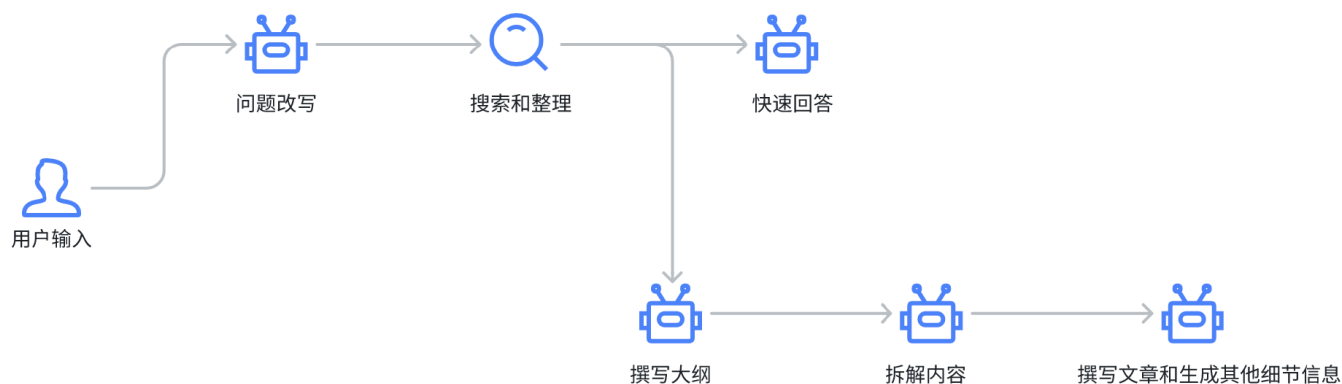
一是先针对问题和参考资料给出一轮快速简答，这样能够快速给出一个比较基础的回答，减少用户的等待时间。

二是针对问题和参考资料，构建内容大纲、拆解子标题、完成段落撰写，最后生成其他需要的内容，比如生成段落配图的提示词。

针对详细内容拆解的过程，有一套方法论，叫做 **PBL (Problem-Based Learning) 教学法**。

PBL 教学法，即“问题导向学习法”，是一种以学生为中心、通过解决真实情境中的复杂问题来促进知识建构与能力培养的教学模式。其核心思想是：将学习置于“问题情境”之中，让学生主动探索、合作交流、自主反思，从而形成深度理解与可迁移的技能。

我们将波波熊学伴的整体工作流完整的节点拆解如下图：



接下来我们就来具体实现其中的每个步骤。

问题改写和选择

首先，我们要实现用户输入后问题的改写选择的具体交互逻辑。在前一节课，我们已经把这块底层逻辑给实现了，现在我们来添加交互细节。

我们打开 Bearbobo Discovery 项目，首先将前一节课的问题改写逻辑添加到项目中。

我们创建 `/lib/prompts/make-question.tpl.ts`，内容如下：

[复制代码](#)


```
1 export default `你是一个儿童科普助手，负责分析孩子的好奇心问题。
2
3 孩子有时候输入的问题不够具体，这时候你需要根据孩子的输入推断出孩子真正想要问的内容，给出三个最可能
4
5 如果你认为孩子问的问题已经足够具体（即已经是一个完整的疑问句，且问题适合孩子问），那么只输出当前这
6
7 # Example
8 当孩子问“火焰”时，孩子可能要问的是：
9
```

```

10 火焰的温度有多高?
11 火焰为什么会有不同的颜色?
12 火焰是如何产生的?
13
14 # Output
15 你输出一个JSON对象, 格式如下:
16
17 {
18     "questions": [
19         {
20             question: "火焰的温度有多高? ",
21             query: ["火焰的温度", "火焰高温特诊"]
22         },
23         {
24             question: "火焰为什么会有不同的颜色? ",
25             query: ["火焰的颜色", "火焰颜色与温度关系"]
26         },
27         {
28             question: "火焰是如何产生的? ",
29             query: ["火焰产生的原理", "火焰燃烧的过程"]
30         }
31     ]
32 }

```

然后创建 `server.ts` , 添加一个 http 接口 `make-question` :

 复制代码

```

1  import * as dotenv from 'dotenv'
2  import express from 'express';
3  import { pipeline } from 'node:stream/promises';
4  import { type ChatConfig, Ling } from '@bearbobo/ling';
5  import makeQuestionPrompt from './lib/prompts/make-question.tpl.ts';
6  import bodyParser from 'body-parser';
7
8  dotenv.config({
9      path: ['.env.local', '.env']
10 })
11
12 const apiKey = process.env.VITE_KIMI_API_KEY as string;
13 const endpoint = process.env.VITE_KIMI_END_POINT as string;
14 const modelName = process.env.VITE_KIMI_MODEL_NAME as string;
15
16 const app = express();
17 const port = 3000;
18
19 app.use(express.json({ limit: '50mb' }));


```

```

20 app.use(bodyParser.json());
21
22 const config: ChatConfig = {
23     model_name: modelName,
24     api_key: apiKey,
25     endpoint,
26     sse: true,
27 };
28
29 app.get('/make-question', async (req, res) => {
30     const question = req.query.question as string;
31
32     // ----- The work flow start -----
33     const ling = new Ling(config);
34     const bot = ling.createBot();
35     bot.addPrompt(makeQuestionPrompt);
36     bot.chat(question);
37
38     ling.close();
39
40     // setting below headers for Streaming the data
41     res.writeHead(200, {
42         'Content-Type': "text/event-stream",
43         'Cache-Control': "no-cache",
44         'Connection': "keep-alive"
45     });
46
47     pipeline((ling.stream as any), res);
48 });
49
50 // 启动服务器
51 app.listen(port, () => {
52     console.log(`Server running on http://localhost:${port}`);
53 });

```

在 `/src/components` 下创建一个 Vue 组建 `MakeQuestion.vue`，内容如下：

 复制代码

```

1 <script setup lang="ts">
2 defineProps({
3     questions: {
4         type: Array,
5         default: [],
6     }
7 });

```

```

8   const emit = defineEmits(['selection'])
9   const selection = (question: string) => {
10     emit('selection', question)
11   }
12 }
13 </script>
14
15 <template>
16   <div class="questions">
17     <div>我猜你想要问的是: </div>
18     <div v-for="(question, index) in questions" :key="index">
19       <div class="question selection" @click="selection(question as string)">{{ q
20     </div>
21   </div>
22 </template>
23
24 <style scoped>
25 .questions {
26   display: flex;
27   flex-direction: row;
28   flex-wrap: wrap;
29 }
30
31 .question {
32   margin: 0 10px;
33   font-weight: bold;
34 }
35
36 .question.selection {
37   cursor: pointer;
38 }
39 </style>

```

这个组件的功能比较简单，就是接收一个问题列表数组，将它展示出来供用户选择。


那么列表数组怎么来呢？就是通过我们请求 `/api/make-question` 接口来获得。

我们修改 App.vue：

```

1 <script setup lang="ts">
2 import { ref, type Ref } from 'vue';
3 import MakeQuestion from './components/MakeQuestion.vue';

```

 复制代码


```

4  const question = ref('天空为什么是蓝色的? ');
5
6  const rewrittenQuestions: Ref<Array<string>> = ref([]);
7
8  const update = async () => {
9    if (!question) return;
10   rewrittenQuestions.value = [];
11   const endpoint = '/api/make-question';
12   const eventSource = new EventSource(`${endpoint}?question=${question.value}`);
13
14   eventSource.addEventListener("message", function (e: any) {
15     let { uri, delta } = JSON.parse(e.data);
16     const matches = uri.match(/questions\/(\d+)\question$/);
17     if (matches) {
18       const index = parseInt(matches[1]);
19       rewrittenQuestions.value[index] = rewrittenQuestions.value[index] || '';
20       rewrittenQuestions.value[index] += delta;
21     }
22   });
23   eventSource.addEventListener('finished', () => {
24     console.log('传输完成');
25     eventSource.close();
26   });
27 }
28 const questionSelected = (question: string) => {
29   console.log('questionSelected', question);
30 }
31 </script>
32
33 <template>
34   <div class="container">
35     <div>
36       <label>输入: </label><input class="input" v-model="question" />
37       <button @click="update">提交</button>
38     </div>
39     <div class="output">
40       <MakeQuestion :questions="rewrittenQuestions" @selection="questionSelected"
41     </div>
42   </div>
43 </template>
44
45 <style scoped>
46 .container {
47   display: flex;
48   flex-direction: column;
49   align-items: center;
50   justify-content: start;
51   width: 100%;
52

```

```
53   height: 100vh;
54   font-size: .85rem;
55 }
56
57 .input {
58   width: 200px;
59 }
60
61 .output {
62   margin-top: 30px;
63   min-height: 300px;
64   width: 100%;
65   text-align: left;
66 }
67 </style>
```

在这里，我们调用 `/api/make-question` 接口，通过 `EventSource` 拿到数据，根据提示词设定的 JSON 数据结构，匹配 `/questions\\/(\\d+)\\/question` 接口数据，将它传给 `MakeQuestion` 组件。核心代码如下：

 复制代码

```
1   eventSource.addEventListener("message", function (e: any) {
2     let { uri, delta } = JSON.parse(e.data);
3     const matches = uri.match(/questions\\/(\\d+)\\/question$/);
4     if (matches) {
5       const index = parseInt(matches[1]);
6       rewrittenQuestions.value[index] = rewrittenQuestions.value[index] || '';
7       rewrittenQuestions.value[index] += delta;
8     }
9   });
```

这样，我们就实现了改写问题并选择的主体逻辑，具体 UI 效果如下：




搜索与快速回答

好了，现在我们实现了问题的改写选择，接下来我们要执行搜索整理资料，并进行简答。

search 的部分我们在上一节课已经封装好了模块，那么现在我们还需要实现具体的搜索和简答 API。

首先创建 `/lib/prompts/quick-answer.tpl.ts`，内容如下：

 复制代码

```
1 export default `# Overall Rules to follow
2 1. Do response in 简体中文.
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 students' learning style preferences
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 你正在模拟一个科学教育家，以<Study Style>的方式快速回答<Student Information>的学生的好奇心
20 1. 学生会给你一个好奇心问题，你以科普的方式对这个问题作出**简明扼要**的回答。
21 2. 你的答案要严谨科学，包含对问题的根本原理解释，但需要以合适学生理解的方式表达，以助于学生快速
22 3. 将回答的内容控制在300字以内，简明扼要，将内容分成1-3个小段，自然的串联起知识点，使用段落叙述
23 4. 回答请用**书面语**，不要有对提问者的称呼，回答形式参考[Example]
24 5. 该学生年龄是 {{age}} 岁，务必用适合学生年龄的能理解的方式回答。
25
26 # Example
27 [为什么焦虑的时候会出汗? ]
28 这是“战斗或逃跑反应”的一部分。在产生“或战或逃”反应时，人体的交感神经系统会释放激素，其中包括肾上腺
29 因此，有一种理论认为，这种出汗是一种利于进化的行为：一人焦虑出汗，其他人会更加警觉，准备好应对任何
30 `
```

你可能会疑惑，为什么我们上面这段提示词有中文也有英文？这个是因为我们产品里有些配置项，那部分提示词我们之前用英文撰写的，这个没有关系，并不影响大模型的理解，如果你自己实现时，改成中文也可以。

这部分提示词中有些配置参数，主要在 Student Information 和 Student Style 两个部分，其中一些配置是用户注册后在个人信息里面维护的，我做了一些简化，用固定值替代了这些配置项，只保留性别 gender 和年龄 age。但为了简化操作便于讲解，我会在 server.ts 中将这部

分内容写死，有兴趣的同学可以在课后自己修改这些配置，看看更改后 AI 输出的内容会有什么不同。

现在我们继续，在 server.ts 中添加 `/quick-answer` 接口：


 复制代码

```
1  ...
2  import quickAnswerPrompt from './lib/prompts/quick-answer.tpl.ts';
3  ...
4
5  app.get('/quick-answer', async (req, res) => {
6      const question = req.query.question as string;
7
8      // ----- The work flow start -----
9      const ling = new Ling(config);
10     const bot = ling.createBot('quick-answer', {}, {
11         response_format: { type: 'text' }
12     });
13     bot.addPrompt(quickAnswerPrompt, {
14         gender: 'female',
15         age: '6',
16     });
17     bot.chat(question);
18
19     ling.close();
20
21     // setting below headers for Streaming the data
22     res.writeHead(200, {
23         'Content-Type': 'text/event-stream',
24         'Cache-Control': 'no-cache',
25         'Connection': 'keep-alive'
26     });
27
28     pipeline((ling.stream as any), res);
29 });
```

在上面的代码中，我们让 `quick-answer` 以文本方式返回，所以将 `response_format` 设为 `text`。

我们将 `quickAnswerPrompt` 传给 `bot`，设置孩子年龄为 6 岁，性别为女孩。这样 AI 会返回更适合 6 岁女孩理解的回答内容。


好了，接下来我们修改 App.vue，在 questionSelected 方法中添加新的逻辑：

 复制代码

```
1 <script setup lang="ts">
2 ...
3 import { marked } from 'marked';
4 ...
5
6 const quickAnswer = ref('');
7 const questionSelected = (question: string) => {
8   // console.log('questionSelected', question);
9   quickAnswer.value = '';
10  const endpoint = '/api/quick-answer';
11  const eventSource = new EventSource(`${endpoint}?question=${question}`);
12  eventSource.addEventListener("message", function (e: any) {
13    let { delta } = JSON.parse(e.data);
14    quickAnswer.value += delta;
15  });
16  eventSource.addEventListener('finished', () => {
17    console.log('传输完成');
18    eventSource.close();
19  });
20 }
21 </script>
22
23 <template>
24 ...
25   <div class="output">
26     <MakeQuestion :questions="rewritdQuestions" @selection="questionSelected"
27     <div v-html="marked.parse(quickAnswer)"></div>
28   </div>
29 ...
30 </template>
```

因为 AI 有可能输出 markdown 格式数据，所以我们这里用 marked 来解析 markdown 文本。

我们需要安装一下 marked 依赖库：

 复制代码

```
1 pnpm i marked
```

这样，我们实现了简答的基础功能。来看一下效果：



启用 RAG

注意到我们这个版本的简答并没有使用 search。如果用户询问一些和时事有关的问题，最好还是先搜索一下资料，否则由于大模型语料库的问题，很可能不能非常好地进行解答。

我们现在修改 server.ts，引入 search 接口：

```
1 ...
2 import { search } from './lib/service/serper.search.ts';
3 ...
4 app.get('/quick-answer', async (req, res) => {
5     const question = req.query.question as string;
6     const query = req.query.query as string;
7     let searchResults = '';
8     if (query) {
9         const queries = query.split(';');
10        const promises = queries.map((query) => search(query));
11
12        searchResults = JSON.stringify(await Promise.all(promises));
13    }
14    // ----- The work flow start -----
15    const ling = new Ling(config);
16    const bot = ling.createBot('quick-answer', {}, {
17        response_format: { type: 'text' }
18    });
19    bot.addPrompt(quickAnswerPrompt, {
20        gender: 'female',
21        age: '6',
22    });
23    if (searchResults) bot.addPrompt(`参考资料:\n${searchResults}`)
24    bot.chat(question);
25
26    ling.close();
27
28    // setting below headers for Streaming the data
29    res.writeHead(200, {
30        'Content-Type': 'text/event-stream',
31        'Cache-Control': 'no-cache',
32        'Connection': 'keep-alive'
33    });
34
35    pipeline((ling.stream as any), res);
36 });
```


我们对前端传入的 query 参数（多个 query 以 ; 连接）进行分割，然后调用 search 返回搜索结果。最终我们讲 searchResults 作为参考资料传给 bot，那么 bot 在回答问题时，就会参考搜索结果内容来回答了。

前端我们也增加处理搜索的逻辑，修改 App.vue 代码：

```
1 ...
2 let queries: string[][] = [];
3
4 const update = async () => {
5   if (!question) return;
6   rewrittenQuestions.value = [];
7   quickAnswer.value = '';
8   queries = [];
9
10  const endpoint = '/api/make-question';
11  const eventSource = new EventSource(`${endpoint}?question=${question.value}`);
12
13  eventSource.addEventListener("message", function (e: any) {
14    let { uri, delta } = JSON.parse(e.data);
15    let matches = uri.match(/questions\/(\d+)\\/question$/);
16    if (matches) {
17      const index = parseInt(matches[1]);
18      rewrittenQuestions.value[index] = rewrittenQuestions.value[index] || '';
19      rewrittenQuestions.value[index] += delta;
20    }
21    matches = uri.match(/questions\/(\d+)\\/query\/(\d+)\$/);
22    if (matches) {
23      const index = parseInt(matches[1]);
24      const queryIndex = parseInt(matches[2]);
25      queries[index] = queries[index] || [];
26      queries[index][queryIndex] = queries[index][queryIndex] || '';
27      queries[index][queryIndex] += delta;
28    }
29  });
30  eventSource.addEventListener('finished', () => {
31    console.log('传输完成');
32    eventSource.close();
33  });
34 }
35
36 const quickAnswer = ref('');
37 const questionSelected = (question: string, index: number) => {
38   quickAnswer.value = '';
39   const query = queries[index].join(';');
40   const endpoint = '/api/quick-answer';
41   const eventSource = new EventSource(`${endpoint}?question=${question}&query=${q
42   eventSource.addEventListener("message", function (e: any) {
43     let { delta } = JSON.parse(e.data);
44     quickAnswer.value += delta;
45   });
46   eventSource.addEventListener('finished', () => {
47     console.log('传输完成');
```

```
48     eventSource.close();
49   });
50 }
51 ...
```


因为我们在 make-question 接口中返回了 query 内容（具体格式你可以回顾 make-question.tpl.ts 的内容），所以我们增加一个 matches 过程，匹配出 `/questions/0/query/0` 这样的 uri。

 复制代码

```
1   matches = uri.match(/questions\/(\d+)\query\/(\d+)\$/);
2   if (matches) {
3     const index = parseInt(matches[1]);
4     const queryIndex = parseInt(matches[2]);
5     queries[index] = queries[index] || [];
6     queries[index][queryIndex] = queries[index][queryIndex] || '';
7     queries[index][queryIndex] += delta;
8   }
9
```

这样我们就可以更新 queries。

然后我们在 questionSelected 中多传一个 index 参数，这里我们通过修改 MakeQuestion.vue 的 selection 事件实现：

 复制代码

```
1 <script setup lang="ts">
2 ...
3 const selection = (question: string, index: number) => {
4   emit('selection', question, index)
5 }
6 </script>
7
8 <template>
9 ...
10   <div v-for="(question, index) in questions" :key="index">
11     <div class="question selection" @click="selection(question as string, index
12   </div>
13 ...
```

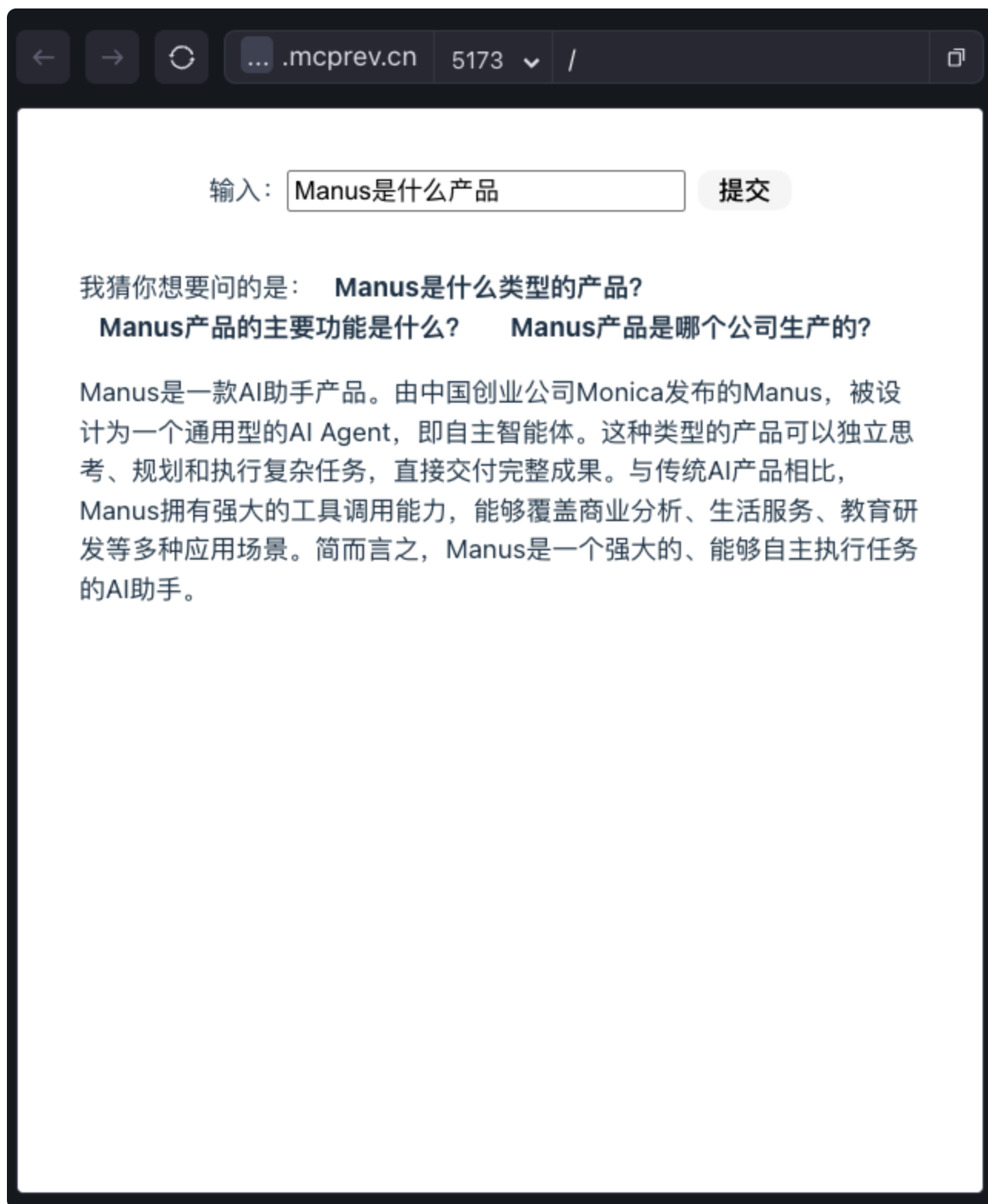

这样当用户选择对应的问题后，我们就可以匹配到 query 词，将它传给 server 处理了。

[📄 复制代码](#)

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

最终，由于有了搜索做支撑，那么当用户问一些比较新的内容时，AI 就能给出相对准确的结果了。

比如下图是当我问 Manus 是什么产品时，AI 给出的结果是后面这样。



到此为止，我们实现了波波熊学伴核心工作流的上半部分，即问题改写->资料搜索和整理->快速回答 这一部分流程。

接下来，我们还有下半部分流程需要并行处理，我把它留到下一节课解决，敬请期待。

要点总结

在这一节里，我们讨论了波波熊学伴产品核心工作流的整体过程，以及上半部分流程，即从问题改写 to 搜索和整理资料再到快速回答的这部分流程。

在这部分流程中，最关键的是对每一个阶段节点的**数据处理和流程控制**，这也是大部分内容类 AI 应用的核心，希望大家通过反复练习，打好基础。

课后练习

为了更好地突出重点，我将波波熊学伴产品中最核心的部分抽取出来讲解，但限于篇幅原因，里面有些细节被我们忽略了，比如服务端我们传给 quick-answer 的提示词，先定了 6 岁女孩，你可以修改这些参数，看看如果孩子的信息改变了，AI 回答的一些内容会不会跟着变化。

另外，在前端，我们也忽略了一些细节。比如如果我问的问题很具体了，那么 AI 在改写问题时只会返回原始问题，这时候我们没必要再用鼠标点击一下进行下一步。如果我问的问题不够具体，那么 AI 会返回三个改写的问题让我选择，而我选择其中一个后，该问题应该高亮或者用其他的 UI 方式标注出来，这样用户才会清晰地知道后续的回答是针对该问题的。这些细节我们都没有去处理，有兴趣的同学，可以修改代码进行研究，然后将结果分享到评论区。

完整的代码在 [🔗 GitHub 仓库](#)。

AI智能总结

1. 本文介绍了波波熊学伴产品整体的核心工作流，包括问题改写和选择的具体交互逻辑，以及搜索与快速回答的实现过程。
2. 通过问题改写和RAG来让大模型减少幻觉，给出质量高的内容，以及通过PBL教学法，将学习置于“问题情境”之中，让学生主动探索、合作交流、自主反思，形成深度理解与可迁移的技能。
3. 作者通过创建相关模板文件和HTTP接口，实现了用户输入后问题的改写选择的具体交互逻辑，以及通过EventSource拿到数据，根据提示词设定的JSON数据结构，匹配接口数据，将它传给MakeQuestion组件，实现了改写问题并选择的主体逻辑。
4. 作者通过封装好的search模块，实现了具体的搜索和简答API，介绍了学生信息和学习风格的相关准则，以及科学教育家快速回答学生的好奇心问题的要求。
5. 实现了波波熊学伴核心工作流的上半部分流程，即`问题改写->资料搜索和整理->快速回答`这一部分流程。
6. 在这部分流程中，最关键的是对每一个阶段节点的数据处理和流程控制，这也是大部分内容类AI应用的核心，希望大家通过反复练习，打好基础。

全部留言 (1)

最新 精选



DARLY
2025-05-20 来自北京

希望能快点更新

