

14 | 项目实战：使用Ling框架重构拍照记单词

月影 · 跟月影学前端智能体开发



你好，我是月影。

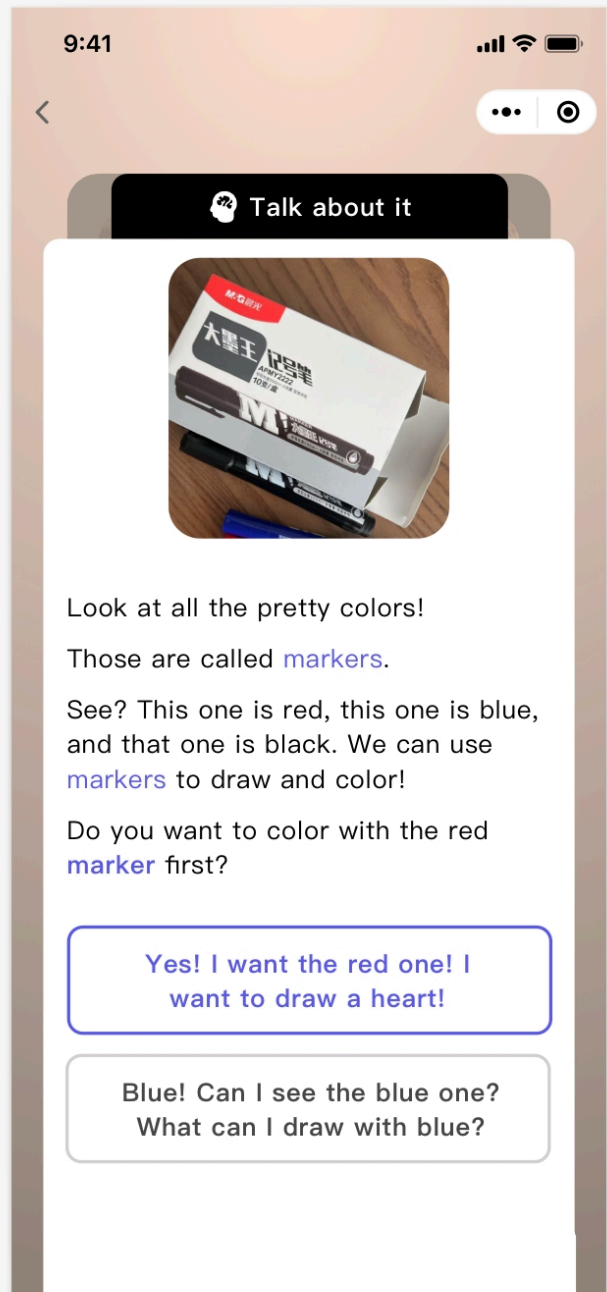
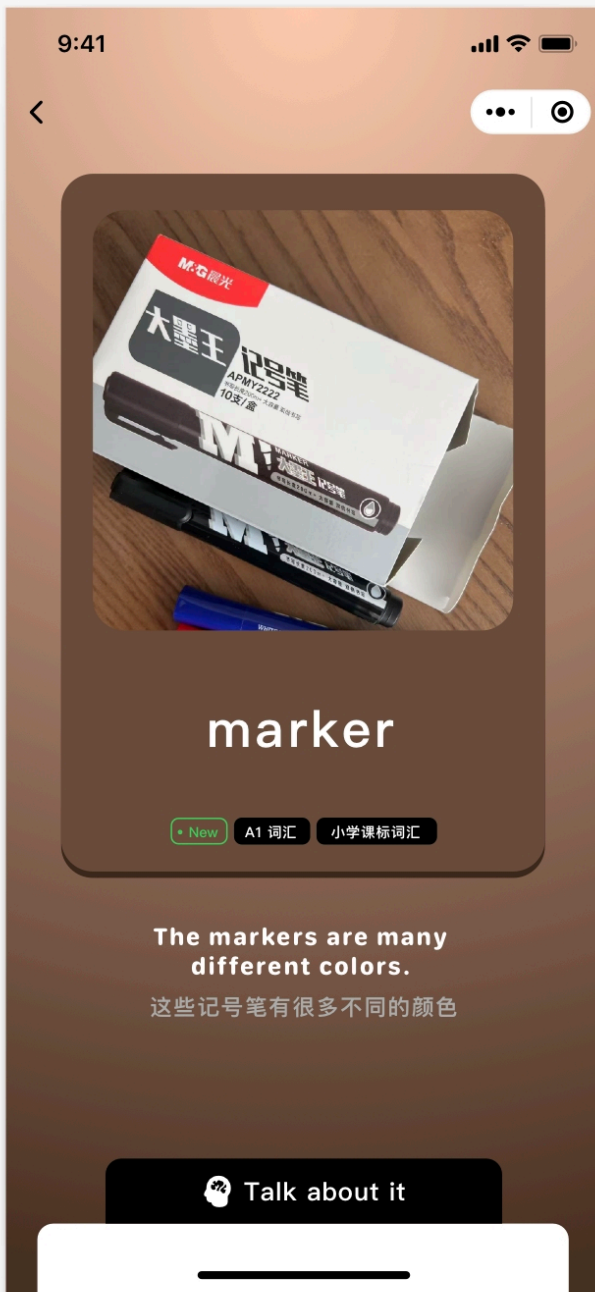
在前面两节课中，我们深入学习了 Ling 框架的实现细节。

在这一节课中，我们就通过实战例子，稍微带你熟悉一下 Ling 的一些高级用法，也让你对 Ling 的能力有一个更直观的认识。

这是第二单元的最后一节必学课，下一章开始，我们就要进入更系统的项目实战单元了。在项目实战单元中，我们会面对更多技术挑战，通过解决这些实际问题，相信那时你也会对 Ling 框架有更深刻的理解。

回顾：拍照记单词

还记得我们之前在第 6 节课创建的应用**拍照记单词**吗？



那时候，我们直接以非流式请求的方式来获取数据，并将数据渲染到页面上。

这么做是有效的，但是从前端用户交互的角度来说，它还是会需要一些用户等待时间。从产品体验的角度来看，效果上不是最好的。




使用 Ling 重构

现在我们来通过对它进行 Ling 框架重构，看看能否有效降低用户等待时间，达到比较好的效果。

我们用 Trae 创建一个项目 Capture the Word Ling。

首先安装依赖项。

 复制代码

```
1 pnpm i jiti dotenv express body-parser @bearbobo/ling jsonuri
2 pnpm i -D @types/express @types/body-parser
```

随后创建配置文件 `.env.local`。

 复制代码

```
1 VITE_KIMI_API_KEY=sk-qi2oJB*****x4p4
2 VITE_KIMI_END_POINT=https://api.moonshot.cn/v1/chat/completions
3 VITE_KIMI_MODEL_NAME=moonshot-v1-8k-vision-preview
4
5 VITE_AUDIO_APP_ID=5934290469
6 VITE_AUDIO_ACCESS_TOKEN=c-LRys*****Ln4N
7 VITE_AUDIO_CLUSTER_ID=volcano_tts
8 VITE_AUDIO_VOICE_NAME=en_female_anna_mars_big tts
```

接着创建 server.ts，内容如下。

 复制代码

```
1 import * as dotenv from 'dotenv'
2 import express from 'express';
3 import { pipeline } from 'node:stream/promises';
4 import { generateAudio } from './lib/audio.ts';
5 import { type ChatConfig, Ling } from '@bearbobo/ling';
6 import userPrompt from './lib/prompt.tpl.ts';
7 import bodyParser from 'body-parser';
8
9 dotenv.config({
10   path: ['.env.local', '.env']
11 })
12
13 const apiKey = process.env.VITE_KIMI_API_KEY as string;
14 const endpoint = process.env.VITE_KIMI_END_POINT as string;
15 const modelName = process.env.VITE_KIMI_MODEL_NAME as string;
```

```
16 const app = express();
17 const port = 3000;
18
19 app.use(express.json({ limit: '50mb' }));
20 app.use(bodyParser.json());
21
22 // SSE 端点
23 app.post('/vision', async (req, res) => {
24     // 设置响应头部
25     res.setHeader('Content-Type', 'text/event-stream');
26     res.setHeader('Cache-Control', 'no-cache');
27     res.setHeader('Connection', 'keep-alive');
28     res.flushHeaders(); // 发送初始响应头
29
30     const imageData = req.body.imageData as string;
31
32     const config: ChatConfig = {
33         model_name: modelName,
34         api_key: apiKey,
35         endpoint,
36     };
37
38     // ----- The work flow start -----
39     const ling = new Ling(config);
40     const bot = ling.createBot();
41     bot.addPrompt(userPrompt);
42     bot.chat([
43         {
44             type: "image_url",
45             image_url: {
46                 "url": imageData,
47             },
48         }, {
49             type: "text",
50             text: userPrompt,
51         }
52     ]);
53
54     bot.on('string-response', ({ uri, delta }) => {
55         console.log('bot string-response', uri, delta);
56     });
57
58     ling.close();
59
60     pipeline((ling.stream as any), res);
61 });
62
63 // 启动服务器
64 app.listen(port, () => {
65     console.log(`Server running on http://localhost:${port}`);
66 });
```

我们用 express 服务器，引入 ling 框架，实现一个 /vision 接口，从前端接收 imageData 参数，然后用它来与 Bot 对话。

[复制代码](#)

```
1 const ling = new Ling(config);
2 const bot = ling.createBot();
3 bot.addPrompt(userPrompt);
4 bot.chat([
5   {
6     type: "image_url",
7     image_url: {
8       url: imageData,
9     },
10  }, {
11    type: "text",
12    text: userPrompt,
13  }
14 ]]);
```

Ling 框架的 stream 属性是一个 Stream 对象（这个属性返回 Ling 创建的 Tube 对象中的 Stream 对象），所以我们可以通过 pipeline 将 express 的 response 对象与它连接在一起，从而实现流式输出。

[复制代码](#)

```
1 pipeline((ling.stream as any), res);
```

接下来，我们还要定义用户提示词。创建 /lib/prompt.tpl.ts，添加我们的提示词：


[复制代码](#)

```
1 export default `
2 分析图片内容，找出最能描述图片的一个英文单词，尽量选择更简单的A1~A2的词汇。
3
4 返回JSON数据：
5 {
6   "representative_word": "图片代表的英文单词",
7   "example_sentence": "结合英文单词和图片描述，给出一个简单的例句",
8   "explanations": ["Look at the...", "", ""], // 用三句话描述图片代表的英文单词，第一句
```

```
9   "explanation_replies": ["根据explanation给出的回复1", "根据explanation给出的回复2"]
10 }
11 `;
```


这样我们就实现了 server 端的主体逻辑。

由于我们这次的 server 是一个 Typescript 文件，所以我们要用 jiti 来启动它，修改 package.json。

 复制代码

```
1 {
2   ...
3   "scripts": {
4     "dev": "jiti server & vite",
5     ...
6   }
```


然后我们别忘了修改 vite.config.js。

 复制代码

```
1 {
2   ...
3   server: {
4     proxy: {
5       '/api': {
6         target: 'http://localhost:3000',
7         changeOrigin: true,
8         rewrite: path => path.replace(/^\/api/, ''),
9       }
10    },
11  }
12 }
```

这样我们就可以通过 `pnpm dev` 启动 server，在 App.vue 中通过访问 `/api/vision` 来对话。

接着我们修改 App.vue。

 复制代码

```
1 <script setup lang="ts">
2 import { ref, type Ref } from 'vue';
3 import PictureCard from './components/PictureCard.vue';
4 import { get, set } from 'jsonuri';
5
6 const word = ref('请上传图片');
7 const audio = ref('');
8 const sentence = ref('');
9
10 const detailExpand = ref(false);
11 const imgPreview = ref('https://res.bearbobo.com/resource/upload/W44yyxvl/upload-');
12
13 const explanations: Ref<string[]> = ref([]);
14 const expReply: Ref<string[]> = ref([]);
15
16 const update = async (imageData: string) => {
17   imgPreview.value = imageData;
18
19   word.value = '分析中...';
20
21   const res = await fetch("/api/vision", {
22     method: 'POST',
23     headers: {
24       'Content-Type': 'application/json',
25     },
26     body: JSON.stringify({
27       imageData,
28     })
29   });
30
31   const reader = res.body!.getReader();
32   const decoder = new TextDecoder();
33   let done = false;
34
35   const data = {
36     representative_word: '',
37     example_sentence: '',
38     explanations: [],
39     explanation_replies: [],
40   };
41
42   while (!done) {
43     const { value, done: doneReading } = await reader.read();
44     done = doneReading;
```



```

45     if (!done) {
46         const content = decoder.decode(value);
47         const lines = content.trim().split('\n');
48         for (const line of lines) {
49             const input = JSON.parse(line);
50             if (input.uri) {
51                 const content = get(data, input.uri);
52                 set(data, input.uri, (content || '') + input.delta);
53                 if (input.uri.endsWith('word')) {
54                     word.value = data.representative_word;
55                 } else if (input.uri.endsWith('sentence')) {
56                     sentence.value = data.example_sentence;
57                 } else if (input.uri.includes('explanations\')) {
58                     explanations.value = [...data.explanations];
59                 } else if (input.uri.includes('explanation_replies')) {
60                     expReply.value = [...data.explanation_replies];
61                 }
62             }
63         }
64     }
65 }
66
67 // const audioUrl = await generateAudio(replyData.example_sentence);
68 // audio.value = audioUrl;
69 };
70
71 const submit = async (imageData: string) => {
72     // console.log(imageData);
73     update(imageData);
74 };
75 </script>
76
77 <template>
78     <div class="container">
79         <PictureCard @update-image="submit" :word="word" :audio="audio" />
80         <div class="output">
81             <div>{{ sentence }}</div>
82             <div class="details">
83                 <button @click="detailExpand = !detailExpand">Talk about it</button>
84                 <div v-if="!detailExpand" class="fold"></div>
85                 <div v-else class="expand">
86                     
87                     <div class="explanation" v-for="item in explanations">
88                         <p>{{ item }}</p>
89                     </div>
90                     <div class="reply" v-for="item in expReply">
91                         <p>{{ item }}</p>
92                     </div>
93                 </div>

```


```
94     </div>
95   </div>
96 </div>
97 </template>
98
99 <style scoped>
100 .container {
101   display: flex;
102   flex-direction: column;
103   align-items: center;
104   justify-content: start;
105   margin: 0;
106   padding: 0;
107   width: 100vw;
108   height: 100vh;
109   font-size: .85rem;
110   background: linear-gradient(180deg, rgb(235, 189, 161) 0%, rgb(71, 49, 32) 100%
111 }
112
113 #selecteImage {
114   display: none;
115 }
116
117 .input {
118   width: 200px;
119 }
120
121 .output {
122   margin-top: 20px;
123   /* min-height: 300px; */
124   width: 80%;
125   text-align: center;
126   font-weight: bold;
127 }
128
129 .preview img {
130   max-width: 100%;
131 }
132
133 button {
134   padding: 0 10px;
135   margin-left: 6px;
136 }
137
138 .details {
139   position: absolute;
140   bottom: 0;
141   left: 50%;
142   transform: translateX(-50%);
```

```
143 }
144
145 .details button {
146     background-color: black;
147     color: white;
148     width: 160px;
149     height: 32px;
150     border-radius: 8px 8px 0 0;
151     border: none;
152     font-size: 12px;
153     font-weight: bold;
154     cursor: pointer;
155 }
156
157 .details .fold {
158     width: 200px;
159     height: 30px;
160     background-color: white;
161     border-top-left-radius: 8px;
162     border-top-right-radius: 8px;
163 }
164
165 .details .expand {
166     width: 200px;
167     height: 88vh;
168     background-color: white;
169     border-top-left-radius: 8px;
170     border-top-right-radius: 8px;
171 }
172
173 .expand img {
174     width: 60%;
175     margin-top: 20px;
176     border-radius: 6px;
177 }
178
179 .expand .explanation {
180     color: black;
181     font-weight: normal;
182 }
183
184 .expand .explanation p {
185     margin: 0 10px 10px 10px;
186 }
187
188 .expand .reply {
189     color: black;
190     font-weight: normal;
191     margin-top: 20px;
```

```
192 }
193
194 .expand .reply p {
195     padding: 4px 10px;
196     margin: 0 10px 10px 10px;
197     border-radius: 6px;
198     border: solid 1px grey;
199 }
200 </style>
```


我们看一下代码的关键部分。

首先我们直接请求 `/api/vision` 。

 复制代码

```
1  const res = await fetch("/api/vision", {
2    method: 'POST',
3    headers: {
4      'Content-Type': 'application/json',
5    },
6    body: JSON.stringify({
7      imageData,
8    })
9  });
```

服务端返回流式输出，我们可以立即对它进行处理。

 复制代码

```
1  const reader = res.body!.getReader();
2  const decoder = new TextDecoder();
3  let done = false;
4
5  const data = {
6    representative_word: '',
7    example_sentence: '',
8    explanations: [],
9    explanation_replies: [],
10 };
11
12 while (!done) {
13   const { value, done: doneReading } = await reader.read();
```

```

14   done = doneReading;
15   if (!done) {
16       const content = decoder.decode(value);
17       const lines = content.trim().split('\n');
18       for (const line of lines) {
19           const input = JSON.parse(line);
20           if (input.uri) {
21               const content = get(data, input.uri);
22               set(data, input.uri, (content || '') + input.delta);
23               if (input.uri.endsWith('word')) {
24                   word.value = data.representative_word;
25               } else if (input.uri.endsWith('sentence')) {
26                   sentence.value = data.example_sentence;
27               } else if (input.uri.includes('explanations\')) {
28                   explanations.value = [...data.explanations];
29               } else if (input.uri.includes('explanation_replys')) {
30                   expReply.value = [...data.explanation_replys];
31               }
32           }
33       }
34   }
35 }

```

UI 组件方面，我们直接复用第 6 节课的 PictureCard 组件即可。

📄 复制代码

```

1  <script setup lang="ts">
2  import { ref } from 'vue';
3  const imgPreview = ref('https://res.bearbobo.com/resource/upload/W44yyxvl/upload-
4
5  const emit = defineEmits(['updateImage']);
6
7  const props = defineProps({
8      word: {
9          type: String,
10         default: '',
11     },
12     audio: {
13         type: String,
14         default: '',
15     }
16 });
17
18 const updateImageData = async (e: Event): Promise<any> => {
19     const file = (e.target as HTMLInputElement).files?.[0];

```

```
20   if (!file) {
21     return;
22   }
23
24   return new Promise((resolve, reject) => {
25     const reader = new FileReader();
26     reader.readAsDataURL(file);
27     reader.onload = () => {
28       const data = reader.result as string;
29       imgPreview.value = data;
30       emit('updateImage', data);
31       resolve(data);
32     };
33     reader.onerror = (error) => {
34       reject(error);
35     };
36   });
37 };
38
39 const playAudio = () => {
40   const audio = new Audio(props.audio);
41   audio.play();
42 }
43 </script>
44
45 <template>
46   <div class="card">
47     <input id="selecteImage" class="input" type="file" accept=".jpg, .jpeg, .png,"
48     <label for="selecteImage" class="upload">
49       
50     </label>
51     <div class="word">{{ props.word }}</div>
52     <div v-if="audio" class="playAudio" @click="playAudio">
53       <img width="20px" src="https://res.bearbobo.com/resource/upload/0mq2HFs8/
54     </div>
55   </div>
56 </template>
57
58 <style scoped>
59 #selecteImage {
60   display: none;
61 }
62 .card {
63   border-radius: 8px;
64   padding: 20px;
65   margin-top: 40px;
66   height: 280px;
67   box-shadow: rgb(63,38,21) 0 3px 0px 0;
68   background-color: rgb(105,78,62);
```

```
69   box-sizing: border-box;
70 }
71 .upload {
72   width: 160px;
73   height: 160px;
74   display: flex;
75   flex-direction: column;
76   align-items: center;
77   justify-content: center;
78 }
79
80 .upload img {
81   width: 100%;
82   height: 100%;
83   object-fit: contain;
84 }
85 .word {
86   margin-top: 20px;
87   font-size: 16px;
88   color: rgb(255,255,255);
89 }
90 .playAudio {
91   margin-top: 16px;
92 }
93
94 .playAudio img {
95   cursor: pointer;
96 }
97 </style>
```

这样我们就完成了应用主体功能，我们先尝试用一下看效果：




和前面的第 6 节课对比，你会发现，由于内容用了流式动态生成的方式，用户的等待时间缩短了很多。

增加语音阅读功能

前面我们只是实现了文字流式输出，并没有实现语音朗读的功能，现在我们增加朗读功能。

首先，我们创建 `/lib/audio.ts`，增加语音模块：

 复制代码

```
1 export const generateAudio = async (text: string) => {
2   const token = process.env.VITE_AUDIO_ACCESS_TOKEN;
3   const appId = process.env.VITE_AUDIO_APP_ID;
4   const clusterId = process.env.VITE_AUDIO_CLUSTER_ID;
5   const voiceName = process.env.VITE_AUDIO_VOICE_NAME;
6
7   const endpoint = 'https://openspeech.bytedance.com/api/v1/tts';
8   const headers = {
9     'Content-Type': 'application/json',
10    Authorization: `Bearer;${token}`,
11  };
12
13  const payload = {
14    app: {
15      appid: appId,
16      token,
17      cluster: clusterId,
18    },
19    user: {
20      uid: 'bearbobo',
21    },
22    audio: {
23      voice_type: voiceName,
24      encoding: 'ogg_opus',
25      compression_rate: 1,
26      rate: 24000,
27      speed_ratio: 1.0,
28      volume_ratio: 1.0,
29      pitch_ratio: 1.0,
30      emotion: 'happy',
31      // language: 'cn',
32    },
33    request: {
34      reqid: Math.random().toString(36).substring(7),
35      text,
36      text_type: 'plain',
37      operation: 'query',
38      silence_duration: '125',
39      with_frontend: '1',
40      frontend_type: 'unitTson',
41      pure_english_opt: '1',
42    },
43  };
44}
```

```
45     const res = await fetch(endpoint, {
46         method: 'POST',
47         headers,
48         body: JSON.stringify(payload),
49     });
50     const data = await res.json();
51
52     if (!data.data) {
53         throw new Error(JSON.stringify(data));
54     }
55     return data.data;
56 }
```

这个模块的内容，我们在第 6 节已经讲过了，这里就不再重复了。

接着我们修改 server.ts：

```
1 ...
2 const audioBuffers: Record<string, Buffer> = {};
3
4 ...
5 bot.on('string-response', ({ uri, delta }) => {
6   if (uri.endsWith('example_sentence')) {
7     ling.handleTask(async () => {
8       const audioData = await generateAudio(delta);
9       const tmpId = Math.random().toString(36).substring(7);
10      audioBuffers[tmpId] = Buffer.from(audioData, 'base64');
11      ling.sendEvent({ uri: 'example_sentence_audio', delta: `/api/audio?`
12    });
13  }
14 });
15 ...
16
17 app.get('/audio', (req, res) => {
18   const id = req.query.id as string;
19   const audioData = audioBuffers[id];
20   if (!audioData) {
21     res.status(404).send('Audio not found');
22     return;
23   }
24   res.setHeader('Content-Type', 'audio/ogg');
25   res.send(audioData);
26 });
```

当 Bot 对象的 `string-response` 事件被触发后，我们判断数据的 uri 是 `example_sentence` 时，delta 的内容就是我们要生成音频的文本。因此我们使用 `ling.handleTask` 方法，创建一个异步处理过程，通过 `await generateAudio(delta)` 处理音频，并拿到音频的 base64 数据。


在上一节课我们提到过，`handlerTask` 处理异步请求，Ling 会判断这些异步请求是否处理完成，等待处理完成后再关闭 Tube 对象，因此我们就可以确保在 Ling 关闭 Tube 对象前将生成的音频发送给前端。

注意这里还有一个细节，因为 `generateAudio` 返回的 Base64 字符串比较大，所以我们不将它的内容直接用 Stream 返回，而是将它缓存起来，通过 `/api/audio?id=tmpId` 的接口进

行访问，并将该接口 url 返回给前端。这是一种返回较大的数据（通常是高清图片、音频、视频等）常用的方法。

在实际真实项目中，我们可能不是像这样直接缓存（这里仅作演示，所以简单处理了），而是将它们上传到 DNS 上，然后将获得的 DNS 的 url 返回给前端。

接着我们修改 App.vue 的代码，增加一个处理分支：

 复制代码

```
1   if (!done) {
2     const content = decoder.decode(value);
3     const lines = content.trim().split('\n');
4     for (const line of lines) {
5       const input = JSON.parse(line);
6       if (input.uri) {
7         const content = get(data, input.uri);
8         set(data, input.uri, (content || '') + input.delta);
9         ...
10      } else if (input.uri.endsWith('example_sentence_audio')) {
11        audio.value = data.example_sentence_audio;
12      }
13    }
14  }
15 }
```

因为 server 通过 `ling.sendEvent(...)` 发送 `{ uri: 'example_sentence_audio', delta: /api/audio?id=${tmpId} }` 给前端，所以前端就可以通过这个属性拿到音频 url。

最终，我们运行应用程序，它也会在单词下方生成播放音频按钮（如下图所示），点击后就可以播放相应的音频效果了。



soup



**I like to make a hearty soup with vegetables
and chicken.**



Talk about it

这样我们就完成了整个拍照记单词项目的重构，新项目的完整代码我也提交到 [🔗 GitHub 项目仓库](#)里，有兴趣的同学可以进一步详细研究。

要点总结

这一节课，我们用 Ling 框架将第 6 节课实现的拍照记单词应用又重构了一遍。对比第 6 节课的版本，你会发现新的版本下，用户不用等待很长时间直到内容生成出来。借助流式输出，用户等待几秒后，内容就会立即开始生成，这样就达到了较好的交互效果。

由于这个例子的 AI 工作流非常简单，仅用了一个大模型节点，所以 Ling 的优势其实并没有完全发挥出来。实际上，在越复杂的业务工作流中使用 Ling，获得的收益往往越大。我们马上要进入第三个单元，通过更加复杂的实战项目来学习如何开发 AI 应用，到时候我们将会继续深入 Ling 的使用，你就会发现它能发挥更大的价值了。

课后练习

在第 6 节课，我就提出让大家尝试给 Talk About 的内容也配上音频，现在有了 Ling，应该更容易做到了。你可以试试看，修改一下代码，增加新的音频功能，把你实现过程中的思考和得到的收获以及最后的结果分享到评论区吧。

AI智能总结

1. Ling框架的实现细节深入学习
2. Ling框架的高级用法实战例子
3. 增加语音阅读功能
4. Ling框架的优势在复杂业务工作流中的应用
5. Ling框架的交互效果改进
6. Ling框架的实际项目应用价值

- 7. Ling框架的异步请求处理方法
- 8. Ling框架的音频功能实现
- 9. Ling框架的课后练习建议

© 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。

全部留言 (1)

最新 精选



GKNick

2025-05-07 来自浙江

安装依赖项这里，写错了，应该是body-parser，而不是 border-parser

作者回复: 嗯嗯，这个是笔误，我修改一下

