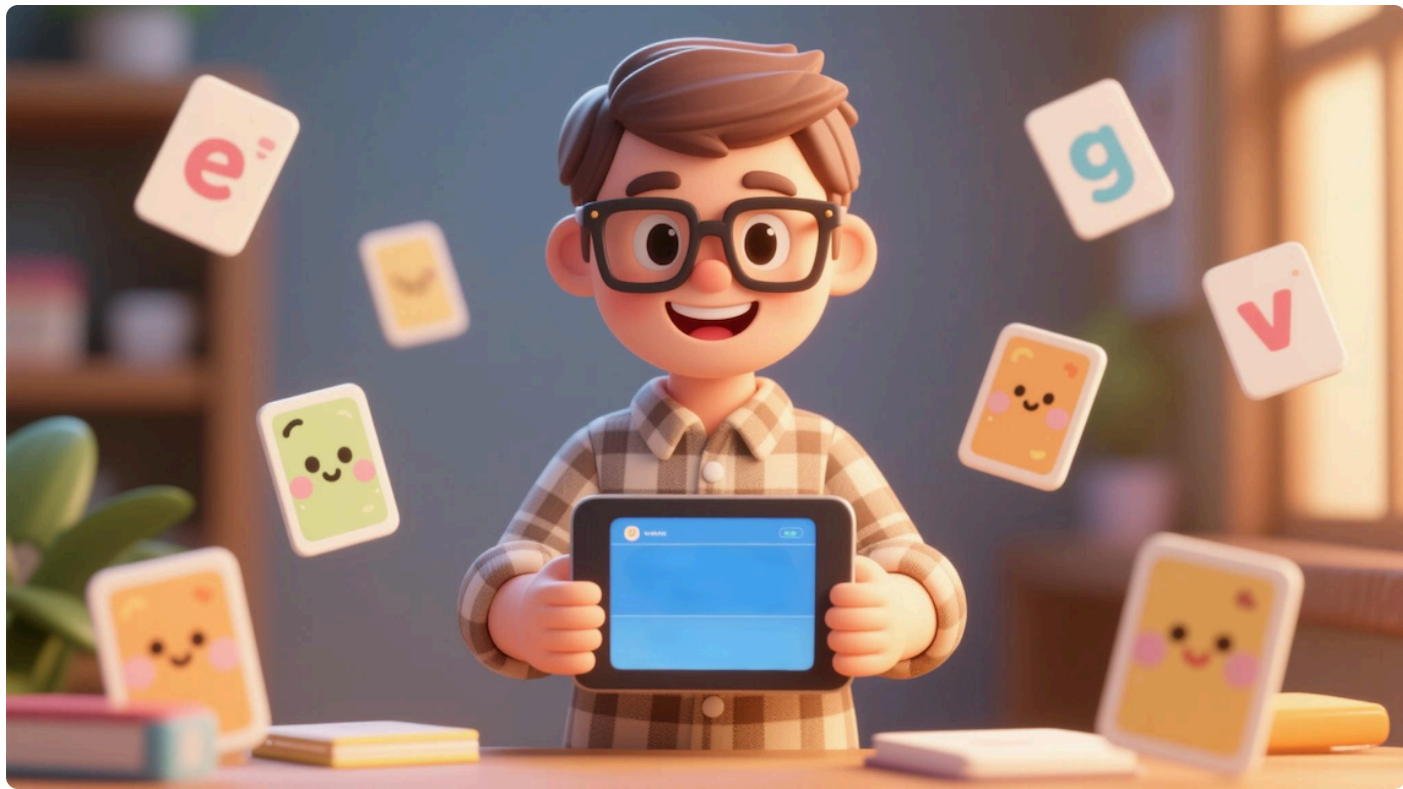


## 06 | 智能体应用实战：拍照记单词

月影 · 跟月影学前端智能体开发



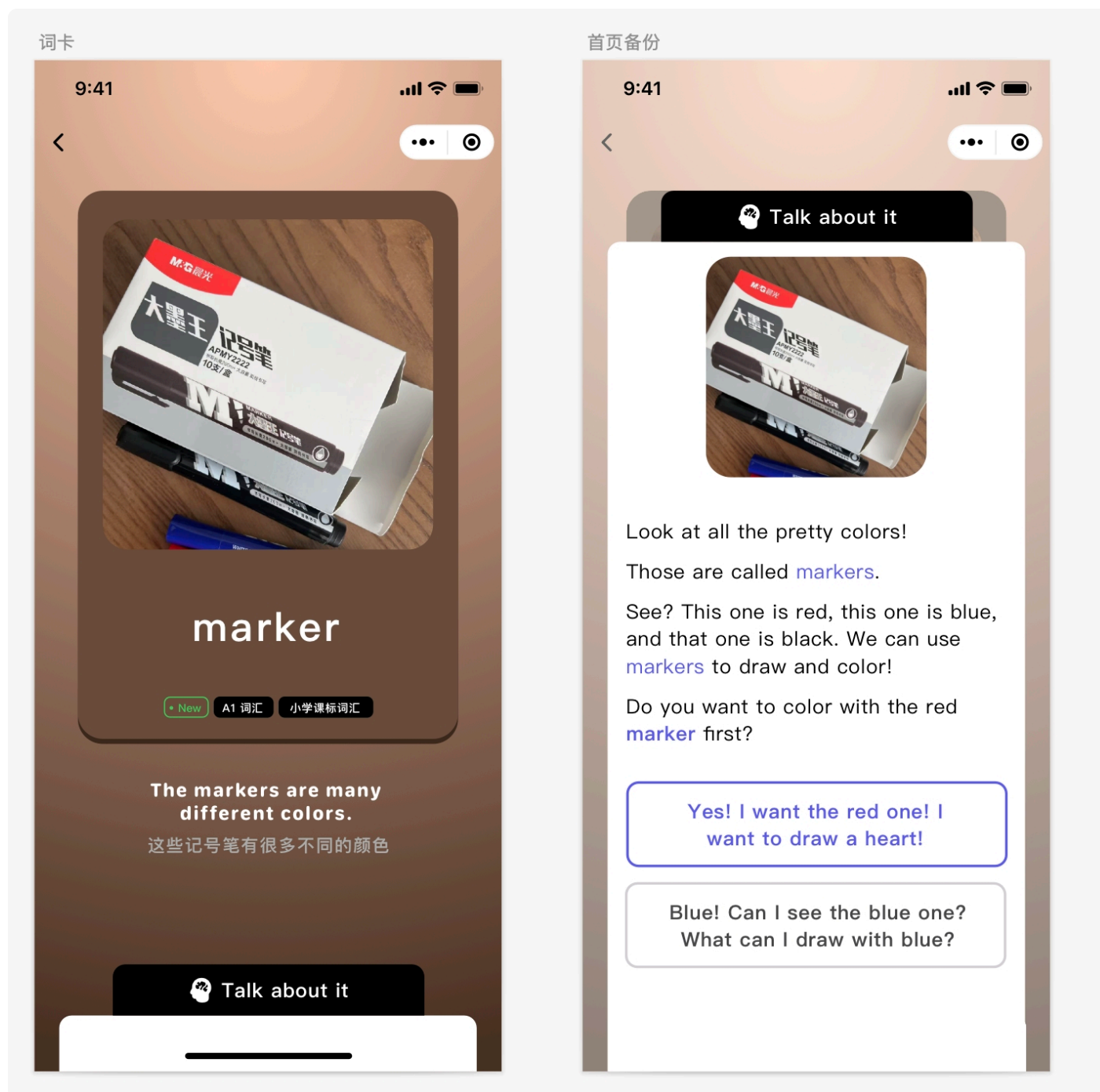
你好，我是月影。

前面我们比较系统地学习了大模型 API 调用和智能体的创建，可能很多同学已经迫不及待地想要将学到的内容运用到具体工作中去了。

那么这节课，我们就来通过一个实战的小应用“拍照记单词”，来具体实践一下如何基于大模型 API 开发 Web 应用。

### 拍照记单词

这个应用的核心功能只有一个，就是用户通过手机拍一张照片，让 AI 识别出照片主体，找到一个最合适的英文单词，生成一张单词卡片。卡片内容包括单词的拼写、读音、例句，以及生成一段对话，以帮助用户快速记忆这个单词和掌握这个单词的用法。



通过需求分析，我们知道这个应用可以通过结合使用两个模型来完成。

首先我们通过上节课使用的视觉模型 moonshot-v1-8k-vision-preview 来解释图片，生成单词、例句和解释，然后通过语音模型来实现朗读功能。整个流程并不复杂，让我们先实现 UI 效果。

我们先用 Trae 创建一个新的项目“Capture the Word”。然后创建.env.local 文件并增加视觉模型配置的 API Key。

```
1 VITE_KIMI_API_KEY=sk-q*****txbp4
```

接着我们实现一个图片上传预览提交的 Vue 组件。

首先创建 `/src/component/PictureCard.vue` 文件，内容如下：


```
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 }
```

```
36   });
37 };
38 </script>
39
40 <template>
41   <div class="card">
42     <input id="selecteImage" class="input" type="file" accept=".jpg, .jpeg, .png,"
43     <label for="selecteImage" class="upload">
44       
45     </label>
46     <div class="word">{{ props.word }}</div>
47     <div v-if="audio" class="playAudio">
48       
2 import PictureCard from './components/PictureCard.vue';
3 ...
4 const word = ref('请上传图片');
5 const audio = ref('');
6 ...
7 </script>
8
9 <template>
10 ...
11   <PictureCard @update-image="submit" :word="word" :audio="audio"/>
12 ...
13 </template>
```

这样就会得到如下效果：



接下来呢，我们修改 App.vue，实现整个界面主体：

 复制代码

```
1 <script setup lang="ts">
2 import { ref } from 'vue';
3 import PictureCard from './components/PictureCard.vue';
4
5 const word = ref('请上传图片');
6 const audio = ref('');
7 const sentence = ref('');
8
9 const detailExpand = ref(false);
10 const imgPreview = ref('https://res.bearbobo.com/resource/upload/W44yyxvl/upload-');
11
12 const explanations = ref([]);
13 const expReply = ref([]);
14
15 const userPrompt = `
16 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。
17
18 返回JSON数据：
19 {
20   "image_discription": "图片描述",
21   "representative_word": "图片代表的英文单词",
22   "example_sentence": "结合英文单词和图片描述，给出一个简单的例句",
23   "explanation": "结合图片解释英文单词，段落以Look at...开头，将段落分句，每一句单独一行，
24   "explanation_replies": ["根据explanation给出的回复1", "根据explanation给出的回复2"]
```

```
25 }
26 `;
27
28 const update = async (imageData: string) => {
29   imgPreview.value = imageData;
30
31   const endpoint = 'https://api.moonshot.cn/v1/chat/completions';
32   const headers = {
33     'Content-Type': 'application/json',
34     Authorization: `Bearer ${import.meta.env.VITE_KIMI_API_KEY}`
35   };
36
37   word.value = '分析中...';
38
39   const response = await fetch(endpoint, {
40     method: 'POST',
41     headers: headers,
42     body: JSON.stringify({
43       model: 'moonshot-v1-8k-vision-preview',
44       messages: [
45         {
46           role: 'user',
47           content: [{
48             type: "image_url",
49             image_url: {
50               "url": imageData,
51             },
52           }, {
53             type: "text",
54             text: userPrompt,
55           }]
56         }
57       ],
58       stream: false,
59     })
60   });
61
62   const data = await response.json();
63   const replyData = JSON.parse(data.choices[0].message.content);
64   word.value = replyData.representative_word;
65   sentence.value = replyData.example_sentence;
66   explanations.value = replyData.explanation.split('\n').filter((item: any) =>
67     expReply.value = replyData.explanation_replies;
68   });
69
70   const submit = async (imageData: string) => {
71     update(imageData);
72   };
73 </script>
```

```

74
75 <template>
76   <div class="container">
77     <PictureCard @update-image="submit" :word="word" :audio="audio"/>
78     <div class="output">
79       <div>{{ sentence }}</div>
80       <div class="details">
81         <button @click="detailExpand = !detailExpand">Talk about it</button>
82         <div v-if="!detailExpand" class="fold"></div>
83         <div v-else class="expand">
84           
85           <div class="explanation" v-for="item in explanations">
86             <p>{{ item }}</p>
87           </div>
88           <div class="reply" v-for="item in expReply">
89             <p>{{ item }}</p>
90           </div>
91         </div>
92       </div>
93     </div>
94   </div>
95 </template>
96
97 <style scoped>
98   .container {
99     display: flex;
100    flex-direction: column;
101    align-items: center;
102    justify-content: start;
103    margin: 0;
104    padding: 0;
105    width: 100vw;
106    height: 100vh;
107    font-size: .85rem;
108    background: linear-gradient(180deg, rgb(235, 189, 161) 0%, rgb(71, 49, 32) 100%
109  }
110
111  #selecteImage {
112    display: none;
113  }
114
115  .input {
116    width: 200px;
117  }
118
119  .output {
120    margin-top: 20px;
121    /* min-height: 300px; */
122    width: 80%;

```



```
123     text-align: center;
124     font-weight: bold;
125 }
126
127 .preview img {
128     max-width: 100%;
129 }
130
131 button {
132     padding: 0 10px;
133     margin-left: 6px;
134 }
135
136 .details {
137     position: absolute;
138     bottom: 0;
139     left: 50%;
140     transform: translateX(-50%);
141 }
142 .details button {
143     background-color: black;
144     color: white;
145     width: 160px;
146     height: 32px;
147     border-radius: 8px 8px 0 0;
148     border: none;
149     font-size: 12px;
150     font-weight: bold;
151     cursor: pointer;
152 }
153 .details .fold {
154     width: 200px;
155     height: 30px;
156     background-color: white;
157     border-top-left-radius: 8px;
158     border-top-right-radius: 8px;
159 }
160
161 .details .expand {
162     width: 200px;
163     height: 88vh;
164     background-color: white;
165     border-top-left-radius: 8px;
166     border-top-right-radius: 8px;
167 }
168 .expand img {
169     width: 60%;
170     margin-top: 20px;
171     border-radius: 6px;
```


```

172 }
173 .expand .explanation {
174     color: black;
175     font-weight: normal;
176 }
177 .expand .explanation p {
178     margin: 0 10px 10px 10px;
179 }
180 .expand .reply {
181     color: black;
182     font-weight: normal;
183     margin-top: 20px;
184 }
185 .expand .reply p {
186     padding: 4px 10px;
187     margin: 0 10px 10px 10px;
188     border-radius: 6px;
189     border: solid 1px grey;
190 }
191 </style>

```

这里，除了一些 UI 细节外，大部分内容是我们前面课程已经学过的。其实关键就是当我们图片更新的时候，通过 PictureCard 的 update-image 事件，将图片的 Base64 发送给 Kimi 的视觉大模型处理。

不过呢，这里需要注意，和我们前面学视觉大模型的时候，让它简单描述图片内容不同，这里，我们使用的是一组提示词，让它输出结构化的 JSON 内容，提示词如下。

 复制代码

```

1  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。
2
3  返回JSON数据：
4  {
5      "image_discription": "图片描述",
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      "explanation": "结合图片解释英文单词，段落以Look at...开头，将段落分句，每一句单独一行，
9      "explanation_replies": ["根据explanation给出的回复1", "根据explanation给出的回复2"
10 }

```

注意我们这里采用了一种结构化输出的技巧，通过 JSON 结构和结构描述，让大模型输出更加合理的内容，这是一种在实际项目中非常实用的技巧，在下一节课中，我们还会进一步详细讲解。

好，现在让我们回到课程代码的部分。

前面的代码我们已经实现了完整的文本内容输出，我们可以测试一下。



我们现在实现了内容文本输出的部分，接下来我们来处理合成语音。

## 合成语音

我们还是基于第四节课讲过的语音模型代码来调整。首先修改.env.local 文件。

 复制代码

```
1 VITE_KIMI_API_KEY=sk-q*****txbp4
2
3 VITE_AUDIO_APP_ID=5934290469
4 VITE_AUDIO_ACCESS_TOKEN=c-L*****Ln4N
5 VITE_AUDIO_CLUSTER_ID=volcano_tts
6 VITE_AUDIO_VOICE_NAME=en_female_anna_mars_big tts
```


我们在其中添加语音配置项，配置 appId、accessToken、clusterId，然后我们将 voiceName 也作为配置项，选择 en\_female\_anna\_mars\_big tts，这是火山引擎语音的音色中比较好听的一个英文人物音色。

接着我们要修改 vite.config.js 文件，添加 server 代理：

 复制代码

```
1 server: {
2   allowedHosts: true,
3   proxy: {
4     '/tts': {
5       target: 'https://openspeech.bytedance.com',
6       changeOrigin: true,
7       rewrite: path => path.replace(/^\/tts/, ''),
8     }
9   },
10 }
```

接着我们来实现一个语音播放的库。在项目中创建文件 /src/lib/audio.ts，内容如下：

 复制代码

```
1 function createBlobURL(base64AudioData: string): string {
2   var byteArrays = [];
3   var byteCharacters = atob(base64AudioData);
4   for (var offset = 0; offset < byteCharacters.length; offset++) {
5     var byteArray = byteCharacters.charCodeAt(offset);
6     byteArrays.push(byteArray);
7   }
8   return URL.createObjectURL(new Blob(byteArrays));
9 }
```

```
7     }
8
9     var blob = new Blob([new Uint8Array(byteArrays)], { type: 'audio/mp3' });
10
11     // 创建一个临时 URL 供音频播放
12     return URL.createObjectURL(blob);
13 }
14
15 export const generateAudio = async (text: string) => {
16     const token = import.meta.env.VITE_AUDIO_ACCESS_TOKEN;
17     const appId = import.meta.env.VITE_AUDIO_APP_ID;
18     const clusterId = import.meta.env.VITE_AUDIO_CLUSTER_ID;
19     const voiceName = import.meta.env.VITE_AUDIO_VOICE_NAME;
20
21     const endpoint = '/tts/api/v1/tts';
22     const headers = {
23         'Content-Type': 'application/json',
24         Authorization: `Bearer;${token}`,
25     };
26
27     const payload = {
28         app: {
29             appid: appId,
30             token,
31             cluster: clusterId,
32         },
33         user: {
34             uid: 'bearbobo',
35         },
36         audio: {
37             voice_type: voiceName,
38             encoding: 'ogg_opus',
39             compression_rate: 1,
40             rate: 24000,
41             speed_ratio: 1.0,
42             volume_ratio: 1.0,
43             pitch_ratio: 1.0,
44             emotion: 'happy',
45             // language: 'cn',
46         },
47         request: {
48             reqid: Math.random().toString(36).substring(7),
49             text,
50             text_type: 'plain',
51             operation: 'query',
52             silence_duration: '125',
53             with_frontend: '1',
54             frontend_type: 'unitTson',
55             pure_english_opt: '1',
```

```


56     },
57   };
58
59   const res = await fetch(endpoint, {
60     method: 'POST',
61     headers,
62     body: JSON.stringify(payload),
63   });
64   const data = await res.json();
65
66   if (!data.data) {
67     throw new Error(JSON.stringify(data));
68   }
69   return createBlobURL(data.data);
70 }

```

这个库的作用，是将指定的文本转换为语音数据，并在客户端生成 BlobURL，这样就可以用 audio 标签播放。

其中细节内容我们在第四节课已经讲过，这里也不再重复，如果有同学还是对代码不了解，可以复习第四节课内容，或者自己动手需改代码通过实践来学习。

接下来，我们继续修改 App.vue 文件，引入 generateAudio 函数，并在 update 方法中使用它。

 复制代码


```

1  ...
2  import { generateAudio } from './lib/audio.ts';
3  ...
4
5  const update = async () => {
6    ...
7
8    const audioUrl = await generateAudio(replyData.example_sentence);
9    audio.value = audioUrl;
10 }

```


这样 App.vue 的主体流程就实现完了，文本模型返回的例句，通过 generateAudio 得到语音播放的 URL，通过 audio 参数传递给 PictureCard 组件。

那么我们还要进一步修改一下 PictureCard 组件，增加播放语音的功能。

 复制代码

```
1 <script setup lang="ts">
2 ...
3 const playAudio = () => {
4   const audio = new Audio(props.audio);
5   audio.play();
6 }
7 </script>
8
9 <template>
10 ...
11   <div v-if="audio" class="playAudio" @click="playAudio">
12     
14 ...
15 </template>
16
17 ...
```

那么最终，完整的 PictureCard 组件代码如下。

 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             
2  import { ref } from 'vue';
3  import PictureCard from './components/PictureCard.vue';
4  import { generateAudio } from './lib/audio.ts';
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/W44yyxv1/upload-
12
13 const explanations = ref([]);

```

```
14 const expReply = ref([]);
15
16 const userPrompt = `
17 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。
18
19 返回JSON数据：
20 {
21   "image_discription": "图片描述",
22   "representative_word": "图片代表的英文单词",
23   "example_sentence": "结合英文单词和图片描述，给出一个简单的例句",
24   "explanation": "结合图片解释英文单词，段落以Look at...开头，将段落分句，每一句单独一行，
25   "explanation_replies": ["根据explanation给出的回复1", "根据explanation给出的回复2"
26 }
27 `;
28
29 const update = async (imageData: string) => {
30   imgPreview.value = imageData;
31
32   const endpoint = 'https://api.moonshot.cn/v1/chat/completions';
33   const headers = {
34     'Content-Type': 'application/json',
35     Authorization: `Bearer ${import.meta.env.VITE_KIMI_API_KEY}`
36   };
37
38   word.value = '分析中...';
39
40   const response = await fetch(endpoint, {
41     method: 'POST',
42     headers: headers,
43     body: JSON.stringify({
44       model: 'moonshot-v1-8k-vision-preview',
45       messages: [
46         {
47           role: 'user',
48           content: [{
49             type: "image_url",
50             image_url: {
51               "url": imageData,
52             },
53           }, {
54             type: "text",
55             text: userPrompt,
56           }
57         ]
58       ],
59       stream: false,
60     })
61   });
62
```

```

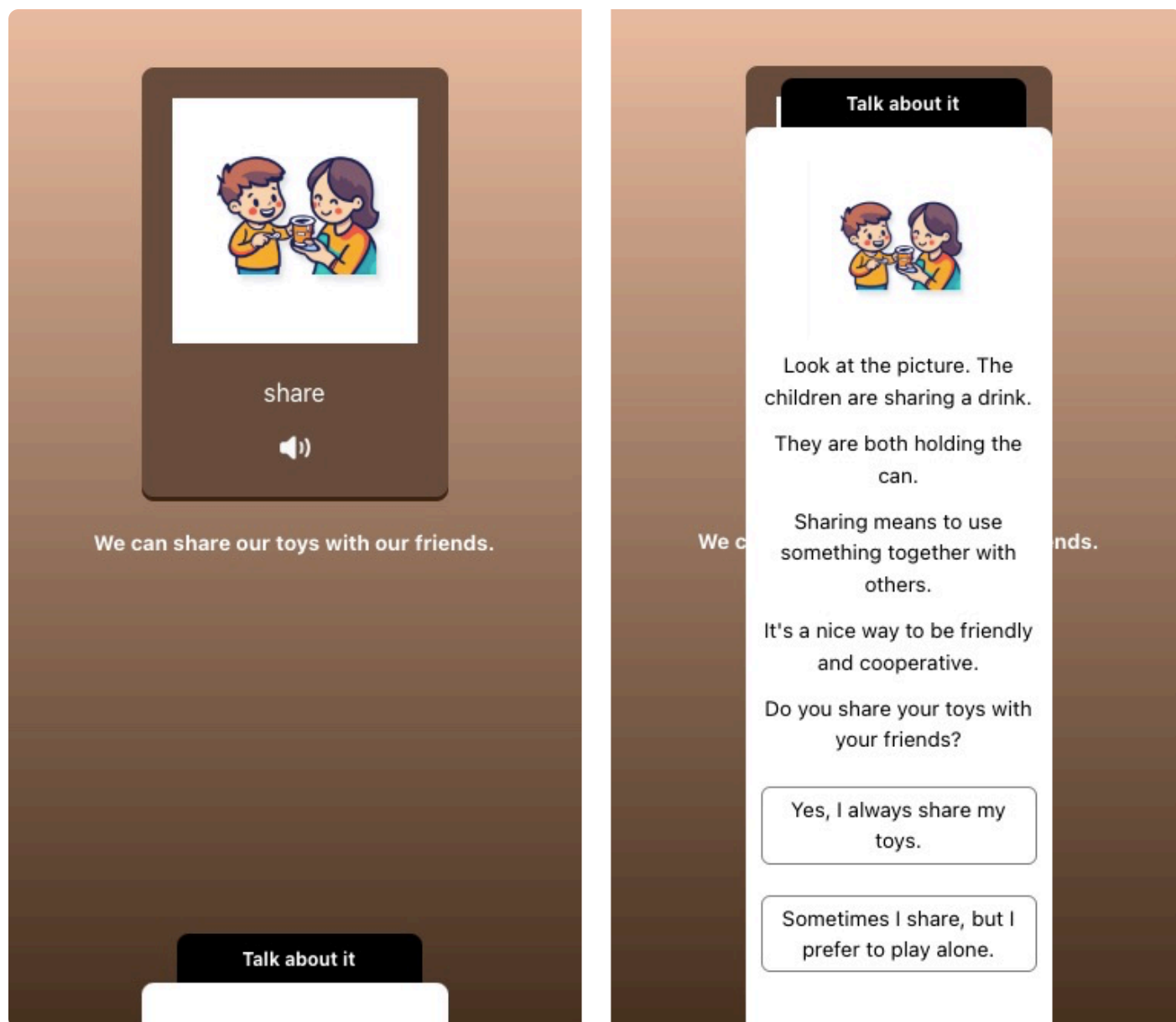
63     const data = await response.json();
64     const replyData = JSON.parse(data.choices[0].message.content);
65     word.value = replyData.representative_word;
66     sentence.value = replyData.example_sentence;
67     explanations.value = replyData.explanation.split('\n').filter((item: any) =>
68     expReply.value = replyData.explanation_replies;
69
70     const audioUrl = await generateAudio(replyData.example_sentence);
71     audio.value = audioUrl;
72 };
73
74 const submit = async (imageData: string) => {
75     // console.log(imageData);
76     update(imageData);
77 };
78 </script>
79
80 <template>
81     <div class="container">
82         <PictureCard @update-image="submit" :word="word" :audio="audio" />
83         <div class="output">
84             <div>{{ sentence }}</div>
85             <div class="details">
86                 <button @click="detailExpand = !detailExpand">Talk about it</button>
87                 <div v-if="!detailExpand" class="fold"></div>
88                 <div v-else class="expand">
89                     
90                     <div class="explanation" v-for="item in explanations">
91                         <p>{{ item }}</p>
92                     </div>
93                     <div class="reply" v-for="item in expReply">
94                         <p>{{ item }}</p>
95                     </div>
96                 </div>
97             </div>
98         </div>
99     </div>
100 </template>
101
102 <style scoped>
103 .container {
104     display: flex;
105     flex-direction: column;
106     align-items: center;
107     justify-content: start;
108     margin: 0;
109     padding: 0;
110     width: 100vw;
111     height: 100vh;

```

```
112     font-size: .85rem;
113     background: linear-gradient(180deg, rgb(235, 189, 161) 0%, rgb(71, 49, 32) 100%
114 }
115
116 #selecteImage {
117     display: none;
118 }
119
120 .input {
121     width: 200px;
122 }
123
124 .output {
125     margin-top: 20px;
126     /* min-height: 300px; */
127     width: 80%;
128     text-align: center;
129     font-weight: bold;
130 }
131
132 .preview img {
133     max-width: 100%;
134 }
135
136 button {
137     padding: 0 10px;
138     margin-left: 6px;
139 }
140
141 .details {
142     position: absolute;
143     bottom: 0;
144     left: 50%;
145     transform: translateX(-50%);
146 }
147
148 .details button {
149     background-color: black;
150     color: white;
151     width: 160px;
152     height: 32px;
153     border-radius: 8px 8px 0 0;
154     border: none;
155     font-size: 12px;
156     font-weight: bold;
157     cursor: pointer;
158 }
159
160 .details .fold {
```

```
161     width: 200px;
162     height: 30px;
163     background-color: white;
164     border-top-left-radius: 8px;
165     border-top-right-radius: 8px;
166 }
167
168 .details .expand {
169     width: 200px;
170     height: 88vh;
171     background-color: white;
172     border-top-left-radius: 8px;
173     border-top-right-radius: 8px;
174 }
175
176 .expand img {
177     width: 60%;
178     margin-top: 20px;
179     border-radius: 6px;
180 }
181
182 .expand .explanation {
183     color: black;
184     font-weight: normal;
185 }
186
187 .expand .explanation p {
188     margin: 0 10px 10px 10px;
189 }
190
191 .expand .reply {
192     color: black;
193     font-weight: normal;
194     margin-top: 20px;
195 }
196
197 .expand .reply p {
198     padding: 4px 10px;
199     margin: 0 10px 10px 10px;
200     border-radius: 6px;
201     border: solid 1px grey;
202 }
203 </style>
```

我们运行一下应用，上传一张图片，效果如下：



点击播放按钮，我们可以听到例句的语音。

你看，这样我们就基本上实现了一个完整的 AI 应用的主体功能！

当然，我们还有很多细节需要处理，其中有 UI 细节，也有程序逻辑中错误分支的细节处理。在实际 AI 应用开发项目中，我们需要处理这些问题，因为我们无法保证这些 API 返回的数据会始终正确，也无法保证 API 服务本身不会出异常。这些都是项目开发中需要重点考虑的，其实不仅仅是 AI 项目，就算是其他 Web 应用项目，我们也是需要考虑容错的。

这些细节问题就不展开一一讲解了，有兴趣的同学，可以基于课程项目代码，自己动手修改细节并不断完善它。

## 要点总结

这一节课，我们第一次进入项目实战，开发一个简单的拍照记单词应用，它的核心功能是上传图片，用视觉大模型的处理能力，生成我们想要的与图片关联的单词例句和单词解释，方便我们学习和记忆单词。

在这个例子中，我们将视觉大模型和大模型语音合成搭配使用，目的是希望同学们能够理解，大模型应用开发往往是通过将不同的模型进行协同工作，最终达到我们想要的效果。

师父领进门，修行在个人。建议你课后实际动手尝试一下，才能真正找到大模型应用开发的感觉。

## 课后练习

拍照记单词应用还有一些细节可以优化，比如功能上，如果我们除了支持例句的语音播放，也希望能支持“Talk about it”中的语音播放，那么应该怎么实现呢？

注意到，我们用大模型生成内容的过程，等待时间相对比较长，有什么办法能够减少用户等待时间吗？请同学们思考一下，将你的想法分享到评论区。

### AI智能总结

1. 应用介绍了一个实战应用“拍照记单词”，通过拍照识别图片主体并生成英文单词卡片。
2. 应用结合使用视觉模型和语音模型来完成，通过结构化的JSON输出内容，实现了更合理的大模型输出。
3. 通过发送图片的Base64给Kimi的视觉大模型处理，实现了图片内容分析和单词识别。
4. 实现了完整的文本内容输出，接下来需要处理合成语音的部分。
5. 调整了语音模型代码，添加了语音配置项，选择了合适的英文人物音色。
6. 实现了一个语音播放的库，将指定的文本转换为语音数据，并在客户端生成BlobURL，实现语音播放的功能。
7. 修改了App.vue文件，引入了生成语音的函数，并在update方法中使用它。
8. 修改了PictureCard组件，增加了播放语音的功能。
9. 实现了一个完整的AI应用的主体功能，上传图片后可以听到例句的语音。



# 全部留言 (2)

最新 精选



Geek\_263db1



2025-04-25 来自北京

如果需要加入自己的知识库呢，如何让模型学习



郑farmer



2025-04-23 来自北京

用扣子搭了一个，就是确实太慢了...

<https://www.coze.cn/s/SPY1rl8x3s4/>

