

09 | 用户体验：如何解决流式传输与JSON结构化的矛盾

月影 · 跟月影学前端智能体开发



你好，我是月影。

在前面的内容里，我们讨论了一些让大模型高质量输出内容的方法。其中让大模型输出 JSON 格式的数据，是一个非常有效且方便的方法。

但是，当我们要进一步改善用户体验，希望通过流式传输减少等待时间时，就会发现 JSON 数据格式本身存在一个问题。

对于从事前端行业的你来说，JSON 应该并不陌生，它是一种**封闭**的数据结构，通常以左花括号“{”开头，右花括号“}”结尾。

封闭的数据结构，意味着一般情况下，前端对 JSON 的解析必须等待 JSON 数据全部传输完成，否则会因为 JSON 数据不完整而导致解析报错。

这就导致一个问题，即使我们在前端用流式获取 JSON 数据，我们也得等待 JSON 完成后才能解析数据并更新 UI，这就让原本流式数据快速响应的特性失效了。

那么有没有办法解决这个问题呢？

JSON 的流式解析

办法是有的。

为了解决这个问题，有些人主张规范大模型的输出，比如采取 NDJSON（Newline-Delimited JSON）的方式，要求大模型输出的内容分为多行，每一行是一个独立的 JSON。但是这么做对大模型的输出进行了限制，不够灵活，而且很可能会影响大模型推理的准确性，有点得不偿失。

另外一些人则使用 JSONStream 库，根据大模型输出的 JSON 配合 JSONStream 使用，这样能一定程度上解决问题，但是也不够通用，必须要事先针对大模型输出的特定结构进行处理，而且只能在 Server 端进行处理，没法直接在前端使用。


我们其实有一个更理想的办法，就是写一个动态解析 JSON 数据流的 parser，然后利用这个 parser 来动态解析返回的数据流。

我在自己实现并开源的 AI 工作流框架 [🔗Ling](#) 中，实现了这个 JSON parser，我们可以将它单独用在我们的项目中。

接下来就让我们通过实践来学习它的用法吧。

首先我们还是用 Trae 创建一个 Vue 项目“JSON Streaming”。

然后配置一下 `.env.local` 文件，这次我们使用 Kimi 大模型。


 复制代码

```
1 VITE_API_KEY=sk-qi2*****x4p4
2 VITE_END_POINT=https://api.moonshot.cn/v1/chat/completions
```

接着我们到 GitHub 的 Ling 仓库，找到 [🔗 /src/parser](#) 目录下的 index.ts 文件并将它复制下来。

我们在 JSON Streaming 项目中创建一个 `/src/lib/json-parser.ts` 文件，将复制的 index.ts 文件内容粘贴过来。

现在有一个问题是，json-parser 依赖库 node:events，我们如果要在前端实践，需要略作改造。我们将文件内容第一行的 `import EventEmitter from 'node:events'` 删掉，并添加如下代码：

 复制代码


```
1 class EventEmitter {
2   private listeners: { [key: string]: ((...args: any[]) => void)[] } = {};
3
4   on(event: string, listener: (...args: any[]) => void) {
5     if (!this.listeners[event]) {
6       this.listeners[event] = [];
7     }
8     this.listeners[event].push(listener);
9   }
10
11   emit(event: string, ...args: any[]) {
12     if (this.listeners[event]) {
13       this.listeners[event].forEach(listener => {
14         listener(...args);
15       });
16     }
17   }
18 }
```

这样我们就完成了 json-parser 的前端改造，解决了依赖库的问题。

json-paser 在对动态数据进行解析的时候，通过 data 事件将增量数据以 {uri, delta} 格式进行传输，所以我们需要将 uri 解析回 JSON 对象，这个操作可以通过 [@jsonuri](#) 库执行。


我们在项目中安装依赖：

```
1 pnpm i jsonuri
```

 复制代码

然后我们修改 App.vue 文件：

```
1 <script setup lang="ts">
2 import { ref } from 'vue';
3 import { JSONParser } from './lib/json-parser';
4 import { set, get } from 'jsonuri';
5
6 const question = ref('狼来了');
7 const content = ref('');
8 const contentParsed = ref({
9   story_instruction: '',
10  the_whole_story_content: '',
11  the_whole_story_translate_to_en: '',
12  lessons: []
13 });
14
15 const systemPrompt = `
16 根据用户输入的主题，用**中文**输出以下JSON格式内容：
17
18 {
19   "story_instruction": "",
20   "the_whole_story_content": "",
21   "the_whole_story_translate_to_en": "",
22   "lessons": []
23 }
24 `;
25
26 const update = async () => {
27   if (!question) return;
28   content.value = "思考中...";
29
30   const endpoint = import.meta.env.VITE_END_POINT;
31   const headers = {
```

 复制代码

```

32     'Content-Type': 'application/json',
33     Authorization: `Bearer ${import.meta.env.VITE_API_KEY}`
34 };
35
36 const response = await fetch(endpoint, {
37     method: 'POST',
38     headers: headers,
39     body: JSON.stringify({
40         model: 'moonshot-v1-8k',
41         messages: [
42             { role: 'system', content: systemPrompt },
43             { role: 'user', content: question.value }
44         ],
45         stream: true,
46     })
47 });
48
49 const reader = response.body?.getReader();
50 const decoder = new TextDecoder();
51 const jsonParser = new JSONParser();
52
53 jsonParser.on('data', ({uri, delta}) => {
54     console.log(uri, delta);
55     const content = get(contentParsed.value, uri);
56     set(contentParsed.value, uri, (content || '') + delta);
57 });
58
59 let done = false;
60 let buffer = '';
61 content.value = '';
62
63 while (!done) {
64     const { value, done: doneReading } = await (reader?.read() as Promise<{ value
65     done = doneReading;
66     const chunkValue = buffer + decoder.decode(value);
67     buffer = '';
68
69     const lines = chunkValue.split('\n').filter((line) => line.startsWith('data:
70
71     for (const line of lines) {
72         const incoming = line.slice(6);
73         if (incoming === '[DONE]') {
74             done = true;
75             break;
76         }
77         try {
78             const data = JSON.parse(incoming);
79             const delta = data.choices[0].delta.content;
80             if (delta) {


```

```
81         content.value += delta;
82         jsonParser.trace(delta);
83     }
84 } catch (ex) {
85     buffer += incoming;
86 }
87 }
88 }
89 }
90 </script>
91
92 <template>
93   <div class="container">
94     <div>
95       <label>输入: </label><input class="input" v-model="question" />
96       <button @click="update">提交</button>
97     </div>
98     <div class="output">
99       <textarea>{{ content }}</textarea>
100      <textarea>{{ contentParsed }}</textarea>
101    </div>
102  </div>
103 </template>
104
105 <style scoped>
106 .container {
107   display: flex;
108   flex-direction: column;
109   align-items: start;
110   justify-content: start;
111   height: 100vh;
112   font-size: .85rem;
113 }
114
115 .input {
116   width: 200px;
117 }
118
119 .output {
120   margin-top: 10px;
121   min-height: 300px;
122   width: 100%;
123   text-align: left;
124 }
125
126 button {
127   padding: 0 10px;
128   margin-left: 6px;
129 }
```

```
130
131 textarea {
132   width: 300px;
133   height: 200px;
134   font-size: 10px;
135 }
136 </style>
```


这个文件内容和之前课程里讲的流式 API 类似，只是其中引入了 JSONParser 和 jsonuri：

```
1 import { JSONParser } from './lib/json-parser';
2 import { set, get } from 'jsonuri';
```

 复制代码

首先我们构建一个业务数据结构：

```
1 const contentParsed = ref({
2   story_instruction: '',
3   the_whole_story_content: '',
4   the_whole_story_translate_to_en: '',
5   lessons: []
6 });
```

 复制代码

给出大模型的系统提示词，采用我们第七节课讲的使用 JSON 输出的技巧：

[📄 复制代码](#)

```
1 const systemPrompt = `
2 根据用户输入的主题，用**中文**输出以下JSON格式内容：
3
4 {
5   "story_instruction": "",
6   "the_whole_story_content": "",
7   "the_whole_story_translate_to_en": "",
8   "lessons": []
9 }
10 `;
```

然后我们在处理请求的时候，创建 JSONParser，利用 JSONParser 来解析数据：

[📄 复制代码](#)

```
1 const jsonParser = new JSONParser();
2
3 jsonParser.on('data', ({uri, delta}) => {
4   console.log(uri, delta);
5   const content = get(contentParsed.value, uri);
6   set(contentParsed.value, uri, (content || '') + delta);
7 });
```

最后在我们从流中获取数据的时候，利用 JSONParser 来动态解析内容即可：

[📄 复制代码](#)

```
1 ...
2   try {
3     const data = JSON.parse(incoming);
4     const delta = data.choices[0].delta.content;
5     if (delta) {
6       content.value += delta;
7       jsonParser.trace(delta);
8     }
9   } catch (ex) {
10     buffer += incoming;
11   }
12 ...
```


我们可以看一下最后的效果：



可以看到，当我们点击提交时，上面的输出框给出的是原始数据，它是不完整的 JSON 数据，我们不能立即使用它。而下面的输入框，始终是保持着完整的 JSON 格式，我们随时可以处理它，用它来更新 UI。

这样的话我们就在客户端实现了基础的 JSON 流式解析。记住这个非常重要的能力，我们后续的实战课程中会反复用到它。

流式 JSON 的 SSE 服务

前面我们讲了在客户端使用 JSON 的动态解析，这样虽然很方便，但不够灵活和强大。

因为通常情况下，我们的工作流可以直接配置在 Node.js 端，不需要通过前端转发。而且 JSONParser 还提供了 string-resolve 的事件，能在 JSON 某个属性动态解析完成时，立即获取完整数据并进行下一步处理，这样能够极大地压榨服务端性能，提升数据响应的及时性。

另外，在服务端执行，我们还可以将数据以 SSE 的方式返回给前端，这样前端使用起来就更加简单了。


我们还是通过一个实战例子来说明。

首先我们还是在 Marscode 上创建一个项目叫做“JSON Streaming SSE”。

这次我们在 src 目录的外边建立一个 lib 目录，添加 json-parser.ts 文件，将 <https://github.com/WeHomeBot/ling/blob/main/src/parser/index.ts> 的内容复制过来。

因为这次我们要在服务端使用，所以我们不用改写文件内容，而且将它放置于 src 目录外边的平级目录下。


然后我们添加 `.env.local` 文件进行配置。

 复制代码

```
1 VITE_API_KEY=sk-qi2oJ*****txbp4
2 VITE_END_POINT=https://api.moonshot.cn/v1/chat/completions
3
4 VITE_AUDIO_APP_ID=5934290469
5 VITE_AUDIO_ACCESS_TOKEN=c-LRysB*****Ln4N
6 VITE_AUDIO_CLUSTER_ID=volcano_tts
7 VITE_AUDIO_VOICE_NAME=en_female_anna_mars_big tts
```


这次我们换一个例子，根据用户场景生成英文例句并转换语音，所以我们不仅仅配置 Kimi 大模型，同时也把火山引擎语音合成服务配置上去。

接下来我们要实现 server.js，首先安装必要的依赖。

 复制代码

```
1 pnpm i dotenv express jsonuri jiti
```

然后创建 server.js 文件，内容如下。

 复制代码

```
1 import * as dotenv from 'dotenv'
2 import express from 'express';
3 import { JSONParser } from './lib/json-parser.ts';
4
5 dotenv.config({
6   path: ['.env.local', '.env']
7 })
8
9 const openaiApiKey = process.env.VITE_API_KEY;
10 const app = express();
11 const port = 3000;
12 const endpoint = process.env.VITE_END_POINT;
13
14 const systemPrompt = `
15 你是一位亲子英语启蒙老师，负责设计家庭英语亲子英语例句。
16 根据用户输入的主题，生成不少于10句英文例句。
17
18 输出以下JSON格式内容：
19 {
```

```
20   "example_sentences": [
21     {
22       "english": "example sentence",
23       "chinese": "例句的中文翻译"
24     },
25     ...
26   ]
27 }
28 `;
29
30 // SSE 端点
31 app.get('/stream', async (req, res) => {
32   // 设置响应头部
33   res.setHeader('Content-Type', 'text/event-stream');
34   res.setHeader('Cache-Control', 'no-cache');
35   res.setHeader('Connection', 'keep-alive');
36   res.flushHeaders(); // 发送初始响应头
37
38   try {
39     // 发送 OpenAI 请求
40     const response = await fetch(
41       endpoint,
42       {
43         method: 'POST',
44         headers: {
45           'Authorization': `Bearer ${openaiApiKey}`,
46         },
47         body: JSON.stringify({
48           model: 'moonshot-v1-8k', // 选择你使用的模型
49           messages: [
50             { role: 'system', content: systemPrompt },
51             { role: 'user', content: req.query.question }
52           ],
53           response_format: { type: "json_object" },
54           stream: true, // 开启流式响应
55         })
56       }
57     );
58
59     if (!response.ok) {
60       throw new Error('Failed to fetch from OpenAI');
61     }
62
63     const reader = response.body.getReader();
64     const decoder = new TextDecoder();
65     const jsonParser = new JSONParser({
66       autoFix: true,
67       onError: (error) => {
68         console.error('JSON Parser Error:', error);
```


```

69         }
70     });
71
72     jsonParser.on('data', (data) => {
73         if (data.uri) res.write(`data: ${JSON.stringify(data)}\n\n`); // 发送
74     });
75
76     let done = false;
77
78     let buffer = '';
79
80     // 读取流数据并转发到客户端
81     while (!done) {
82         const { value, done: doneReading } = await reader.read();
83         done = doneReading;
84         const chunkValue = buffer + decoder.decode(value, { stream: true });
85         buffer = '';
86
87         // 按行分割数据, 每行以 "data: " 开头, 并传递给客户端
88         const lines = chunkValue.split('\n').filter(line => line.trim() && li
89         for (const line of lines) {
90             const incoming = line.slice(6);
91             if (incoming === '[DONE]') {
92                 done = true;
93                 break;
94             }
95             try {
96                 const data = JSON.parse(incoming);
97                 const delta = data.choices[0].delta.content;
98                 jsonParser.trace(delta);
99                 // if (delta) res.write(`data: ${delta}\n\n`); // 发送数据到客
100             } catch (ex) {
101                 buffer += incoming;
102             }
103         }
104     }
105
106     res.write('event: end\n'); // 发送结束事件
107     res.write('data: [DONE]\n\n'); // 通知客户端数据流结束
108     res.end(); // 关闭连接
109
110     } catch (error) {
111         console.error('Error fetching from OpenAI:', error);
112         res.write('data: Error fetching from OpenAI\n\n');
113         res.end();
114     }
115 });
116
117 // 启动服务器

```


```
118 app.listen(port, () => {
119     console.log(`Server running on http://localhost:${port}`);
120 });
```

根据上面的代码，我们的系统提示词如下：

 复制代码

```
1 你是一位亲子英语启蒙老师，负责设计家庭英语亲子英语例句。
2 根据用户输入的主题，生成不少于10句英文例句。
3
4 输出以下JSON格式内容：
5 {
6     "example_sentences": [
7         {
8             "english": "example sentence",
9             "chinese": "例句的中文翻译"
10        },
11        ...
12    ]
13 }
```


大模型输出的 JSON 内容，我们通过 jsonParser 进行处理，发送给客户端。

 复制代码

```
1 jsonParser.on('data', (data) => {
2     if (data.uri) res.write(`data: ${JSON.stringify(data)}\n\n`); // 发送数据到
3     });
```

注意我们的 JSONParser 是 Typescript 写的，而 server.js 是用 JS，所以我们前面安装了 jiti 库，它可以让我们混合运行 TS 和 JS 的服务，我们只要执行 `jiti server` 就可以启动 server。

别忘了配置 vite.config.js，转发 server 的接口：

 复制代码


```
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  },

```

接着我们改写客户端的 App.vue，内容如下：

 复制代码


```

1 <script setup lang="ts">
2 import { ref } from 'vue';
3 import { set, get } from 'jsonuri';
4
5 const question = ref('起床');
6 const content = ref({
7   example_sentences: [],
8 });
9
10 const update = async () => {
11   if (!question) return;
12
13   const endpoint = '/api/stream';
14
15   const eventSource = new EventSource(`${endpoint}?question=${question.value}`);
16   eventSource.addEventListener("message", function (e: any) {
17     const { uri, delta } = JSON.parse(e.data);
18     const str = get(content.value, uri);
19     set(content.value, uri, (str || '') + delta);
20   });
21   eventSource.addEventListener('end', () => {
22     console.log('传输完成');
23     eventSource.close();
24   });
25 }
26 </script>
27
28 <template>
29   <div class="container">
30     <div>
31       <label>输入: </label><input class="input" v-model="question" />

```

```
32     <button @click="update">提交</button>
33 </div>
34 <div class="output">
35     <div v-for="sentence in content.example_sentences as any" :key="sentence.en
36         <h3>{{ sentence.english }} </h3>
37         <p>{{ sentence.chinese }} </p>
38     </div>
39 </div>
40 </div>
41 </template>
42
43 <style scoped>
44 .container {
45     display: flex;
46     flex-direction: column;
47     align-items: start;
48     justify-content: start;
49     height: 100vh;
50     font-size: .85rem;
51 }
52
53 .input {
54     width: 200px;
55 }
56
57 .output {
58     margin-top: 10px;
59     min-height: 300px;
60     width: 100%;
61     text-align: left;
62 }
63
64 button {
65     padding: 0 10px;
66     margin-left: 6px;
67 }
68
69 textarea {
70     width: 300px;
71     height: 200px;
72     font-size: 10px;
73 }
74 h3, h3+p {
75     margin: 0;
76     padding: 0;
77 }
78 </style>
```


在 EventSource 中，我们使用 jsonuri 处理服务端返回的数据。

 复制代码

```
1  eventSource.addEventListener("message", function (e: any) {  
2      const { uri, delta } = JSON.parse(e.data);  
3      const str = get(content.value, uri);  
4      set(content.value, uri, (str || '') + delta);  
5  });
```

这样我们就实现了基础的流式 JSON 传输，我们可以看一下实际的效果：



并行处理语音合成

接下来，我们要为英文内容合成语音，这个工作在服务端完成。我们可以利用 JSONParser 的 string-resolve 数据，及时地并行处理语音转换事件。

首先我们在 lib 目录下添加文件 audio.js，内容如下：

```
1 export const generateAudio = async (text) => {
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 }

```

这个文件的原理，我们之前的课程已经讲了很多，这里就不再赘述。

接着我们修改 server.js 文件，引入 generateAudio 进行处理。

 复制代码

```


1  import { generateAudio } from './lib/audio.js';
2  ...
3
4  jsonParser.on('string-resolve', ({ uri, delta }) => {
5    if (uri.includes('english')) {
6      const task = generateAudio(delta);
7      audioPromises.push(task);
8      task.then((base64data) => {
9        const audioUri = uri.replace('english', 'audio');
10       res.write(`data: ${JSON.stringify({ uri: audioUri, delta: base64dat
11     });
12   }
13 });
14 ...
15
16 await Promise.all(audioPromises); // 等待音频数据结束
17
18 res.write('event: end\n'); // 发送结束事件
19 res.write('data: [DONE]\n\n'); // 通知客户端数据流结束
20 res.end(); // 关闭连接
21 ...

```

我们在 jsonParser 的 `string-resolve` 事件中，判断 uri 是否包含 english。若是，则说明当前 delta 内容是完整的英文例句，这时我们将它发送给 generateAudio 异步处理。

注意，由于这是**异步过程**，所以我们需要等待这些音频合成过程结束后才可以关闭，因此我们将异步任务放到 `audioPromises` 列表中，通过 `await Promise.all(audioPromises);` 来等待所有的音频处理结束。

最后，我们改写客户端 `App.vue`，添加音频数据处理和播放部分：

 复制代码

```
1 <script setup lang="ts">
2 ...
3
4 function playAudio(audio: string) {
5   const audioElement = new Audio(audio);
6   audioElement.play();
7 }
8
9 function createBlobURL(base64AudioData: string): string {
10   var byteArrays = [];
11   var byteCharacters = atob(base64AudioData);
12   for (var offset = 0; offset < byteCharacters.length; offset++) {
13     var byteArray = byteCharacters.charCodeAt(offset);
14     byteArrays.push(byteArray);
15   }
16
17   var blob = new Blob([new Uint8Array(byteArrays)], { type: 'audio/mp3' });
18
19   // 创建一个临时 URL 供音频播放
20   return URL.createObjectURL(blob);
21 }
22
23 ...
24 eventSource.addEventListener("message", function (e: any) {
25   let { uri, delta } = JSON.parse(e.data);
26   if (uri.includes('audio')) {
27     delta = createBlobURL(delta);
28   }
29   const str = get(content.value, uri);
30   set(content.value, uri, (str || '') + delta);
31 });
32
33 ...
34 </script>
35
36 <template>
37 ...
38   <div v-for="sentence in (content.example_sentences as any)" :key="sentence.
```

```
39     <h3>{{ sentence.english }}
40      {
33   // 设置响应头部
34   res.setHeader('Content-Type', 'text/event-stream');
35   res.setHeader('Cache-Control', 'no-cache');
36   res.setHeader('Connection', 'keep-alive');
37   res.flushHeaders(); // 发送初始响应头
38
39   try {
40     // 发送 OpenAI 请求
41     const response = await fetch(
42       endpoint,
43       {
44         method: 'POST',
45         headers: {
46           'Authorization': `Bearer ${openaiApiKey}`,
47         },
48         body: JSON.stringify({
49           model: 'moonshot-v1-8k', // 选择你使用的模型

```



```

50         messages: [
51             { role: 'system', content: systemPrompt },
52             { role: 'user', content: req.query.question }
53         ],
54         response_format: { type: "json_object" },
55         stream: true, // 开启流式响应
56     })
57 }
58 );
59
60 if (!response.ok) {
61     throw new Error('Failed to fetch from OpenAI');
62 }
63
64 const reader = response.body.getReader();
65 const decoder = new TextDecoder();
66 const jsonParser = new JSONParser({
67     autoFix: true,
68     onError: (error) => {
69         console.error('JSON Parser Error:', error);
70     }
71 });
72
73 const audioPromises = [];
74
75 jsonParser.on('data', (data) => {
76     if (data.uri) res.write(`data: ${JSON.stringify(data)}\n\n`); // 发送
77 });
78 jsonParser.on('string-resolve', ({ uri, delta }) => {
79     if (uri.includes('english')) {
80         const task = generateAudio(delta);
81         audioPromises.push(task);
82         task.then((base64data) => {
83             const audioUri = uri.replace('english', 'audio');
84             res.write(`data: ${JSON.stringify({ uri: audioUri, delta: base64data })}`);
85         });
86     }
87 });
88
89 let done = false;
90
91 let buffer = '';
92
93 // 读取流数据并转发到客户端
94 while (!done) {
95     const { value, done: doneReading } = await reader.read();
96     done = doneReading;
97     const chunkValue = buffer + decoder.decode(value, { stream: true });
98     buffer = '';


```

```

99
100 // 按行分割数据, 每行以 "data: " 开头, 并传递给客户端
101 const lines = chunkValue.split('\n').filter(line => line.trim() && li
102 for (const line of lines) {
103     const incoming = line.slice(6);
104     if (incoming === '[DONE]') {
105         done = true;
106         break;
107     }
108     try {
109         const data = JSON.parse(incoming);
110         const delta = data.choices[0].delta.content;
111         jsonParser.trace(delta);
112         // if (delta) res.write(`data: ${delta}\n\n`); // 发送数据到客
113     } catch (ex) {
114         buffer += incoming;
115     }
116 }
117 }
118
119 await Promise.all(audioPromises); // 等待音频数据结束
120
121 res.write('event: end\n'); // 发送结束事件
122 res.write('data: [DONE]\n\n'); // 通知客户端数据流结束
123 res.end(); // 关闭连接
124
125 } catch (error) {
126     console.error('Error fetching from OpenAI:', error);
127     res.write('data: Error fetching from OpenAI\n\n');
128     res.end();
129 }
130 });
131
132 // 启动服务器
133 app.listen(port, () => {
134     console.log(`Server running on http://localhost:${port}`);
135 });

```

App.vue

 复制代码

```

1 <script setup lang="ts">
2 import { ref } from 'vue';
3 import { set, get } from 'jsonuri';
4
5 const question = ref('起床');

```

```

6  const content = ref({
7    example_sentences: [],
8  });
9
10 function playAudio(audio: string) {
11   const audioElement = new Audio(audio);
12   audioElement.play();
13 }
14
15 function createBlobURL(base64AudioData: string): string {
16   var byteArrays = [];
17   var byteCharacters = atob(base64AudioData);
18   for (var offset = 0; offset < byteCharacters.length; offset++) {
19     var byteArray = byteCharacters.charCodeAt(offset);
20     byteArrays.push(byteArray);
21   }
22
23   var blob = new Blob([new Uint8Array(byteArrays)], { type: 'audio/mp3' });
24
25   // 创建一个临时 URL 供音频播放
26   return URL.createObjectURL(blob);
27 }
28
29 const update = async () => {
30   if (!question) return;
31
32   const endpoint = '/api/stream';
33
34   const eventSource = new EventSource(`${endpoint}?question=${question.value}`);
35   eventSource.addEventListener("message", function (e: any) {
36     let { uri, delta } = JSON.parse(e.data);
37     if (uri.includes('audio')) {
38       delta = createBlobURL(delta);
39     }
40     const str = get(content.value, uri);
41     set(content.value, uri, (str || '') + delta);
42   });
43   eventSource.addEventListener('end', () => {
44     console.log('传输完成');
45     eventSource.close();
46   });
47 }
48 </script>
49
50 <template>
51   <div class="container">
52     <div>
53       <label>输入: </label><input class="input" v-model="question" />
54       <button @click="update">提交</button>

```

```
55     </div>
56     <div class="output">
57         <div v-for="sentence in (content.example_sentences as any)" :key="sentence."
58             <h3>{{ sentence.english }}
59                 <img v-if="sentence.audio" width="20px"
60                     @click="playAudio(sentence.audio)"
61                     src="https://res.bearbobo.com/resource/upload/9nZenvln/playAudio-l42l
62             </h3>
63             <p>{{ sentence.chinese }} </p>
64         </div>
65     </div>
66 </div>
67 </template>
68
69 <style scoped>
70 .container {
71     display: flex;
72     flex-direction: column;
73     align-items: start;
74     justify-content: start;
75     height: 100vh;
76     font-size: .85rem;
77 }
78
79 .input {
80     width: 200px;
81 }
82
83 .output {
84     margin-top: 10px;
85     min-height: 300px;
86     width: 100%;
87     text-align: left;
88 }
89
90 button {
91     padding: 0 10px;
92     margin-left: 6px;
93 }
94
95 textarea {
96     width: 300px;
97     height: 200px;
98     font-size: 10px;
99 }
100
101 h3,
102 h3+p {
103     margin: 0;
```

```
104 padding: 0;
105 }
106
107 h3 img {
108   cursor: pointer;
109 }
110 </style>
```

整个项目的代码我也提交到了 Github 上，有兴趣的同学可以访问 https://github.com/akira-cn/frontend-dev-large-model-era/tree/main/json_streaming_sse 进一步研究。

要点总结

这节课，我们了解了 JSON 的流式解析基本原理和方法。通过两个实战例子，分别学习了如何在客户端和服务端动态解析 JSON 和实时处理数据流。

实际上**结构化 JSON 数据的流式处理**，是我们实现快速实时响应的 AI 应用非常重要的基础，希望大家能够多多练习，牢固掌握这一技能，后续我们在综合项目实战中，还会进一步使用并深入探索。

课后练习

1. 注意到我们的第二个例子，server 端创建 JSONParser 对象时，传入了参数 autoFix，它的作用是什么？你可以自己实验一下。
2. 仔细阅读 JSONParser 代码，理解一下 data 和 string-resolve 事件的区别，回答为什么我们在语音合成的时候，要在 string-resolve 事件里处理？

你可以将你的答案或者疑问分享到评论区，我们一同交流探讨。

AI智能总结

1. JSON数据格式在前端应用中存在解析等待时间过长的问題，影响用户体验。
2. 解决JSON数据流式传输问题的方法包括采用NDJSON格式、使用JSONStream库，以及自行实现动态解析JSON数据流的parser。
3. 通过自行实现动态解析JSON数据流的parser，可以在前端实现基础的JSON流式解析，提高用户体验。

4. 通过实践学习JSON流式解析的用法，可以在前端项目中应用JSONParser和jsonuri来动态解析数据流，实现流式数据的快速响应和更新UI。
5. 通过JSON流式解析，可以实现在客户端处理不完整的JSON数据，随时更新UI，提高用户体验。
6. 通过两个实战例子，分别学习了如何在客户端和服务端动态解析JSON和实时处理数据流。
7. 结构化JSON数据的流式处理是实现快速实时响应的AI应用非常重要的基础。
8. JSONParser的参数autoFix的作用是什么，需要进行实验验证。
9. 理解data和string-resolve事件的区别，以及为什么在语音合成的时候要在string-resolve事件里处理。
10. 在综合项目实战中，将进一步使用并深入探索JSON流式解析的应用。

© 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。