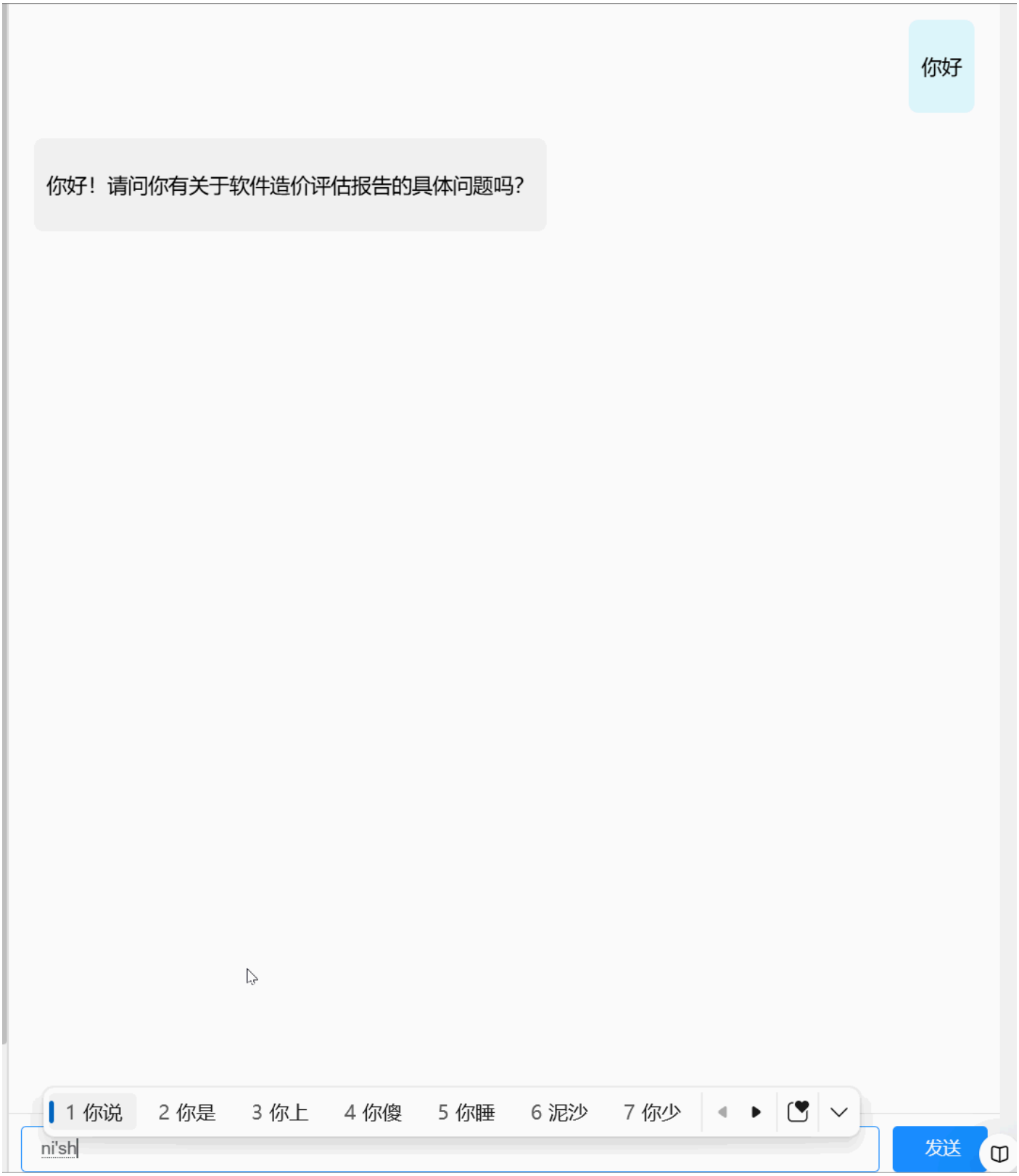
实现一个简易的AI流式对话，模拟ChatGPT（SpringBoot+Vue2）

效果图

二话不说,先上效果,**后续的代码可以直接使用,复现效果图的功能**😊



实现

我们如何实现一个AI流式输出？目前的AI对话接口大多都附带有流式输出接口，但是后端接收到的数据是流式的，我们仍**需要处理数据流并返回给前端，让前端显示流式的效果。**

后端

Controller层

我们使用长轮询获取队列的消息，前端先发送sendMsg请求，后续通过长轮询请求chat接口

```
@RestController
public class LongPollingController{
        // 当前端我们发送一条消息，通知后端调用AI接口回复消息
    @PostMapping("/sendMsg")
    public void receiveMes(@RequestBody MsgReq msgReq) throws NoApiKeyException,
InputRequiredException {
        MessageUtil.streamCall(msgReq.content);
    }

    // 前端我们发送一条消息后，监听长轮询请求，直到队列中有消息
    @GetMapping("/chat")
    public Response handleLongPolling() throws InterruptedException,
NoApiKeyException, InputRequiredException {
        Response message = MessageUtil.getQueue().poll();  // 如果有消息，直接返回；
如果没有，则阻塞直到有消息
        if (message == null) {
            message = MessageUtil.getQueue().take();  // 这里会阻塞直到有消息或超时
        }
        return message;  // 返回消息
    }
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18

实体类

创建两个实体类封装请求和响应

```
@NoArgsConstructor
@AllArgsConstructor
@Data
public class Response {
        // 这个isEnd后面会解释
    boolean isEnd;
    String content;

}

@NoArgsConstructor
@AllArgsConstructor
@Data
public class MsgReq {
    String content;
}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18

MessageUtil工具类

- 我们这里的AI接口使用阿里云百炼API，使用其他的服务实现也是类似，当我们调用百炼API的streamCall接口并返回Flowable数据时，我们后端控制台获取到的数据是这样的：

```
requestId: 6ec84d07-e933-9c88-a53a-d1a00bfd734d, text: 你好！, finishReason: null
requestId: 6ec84d07-e933-9c88-a53a-d1a00bfd734d, text: 请问, finishReason: null
requestId: 6ec84d07-e933-9c88-a53a-d1a00bfd734d, text: 你, finishReason: null
requestId: 6ec84d07-e933-9c88-a53a-d1a00bfd734d, text: 有关于软件造价, finishReason: null
requestId: 6ec84d07-e933-9c88-a53a-d1a00bfd734d, text: 评估报告的具体问题, finishReason: null
requestId: 6ec84d07-e933-9c88-a53a-d1a00bfd734d, text: 吗？我可以帮助, finishReason: null
requestId: 6ec84d07-e933-9c88-a53a-d1a00bfd734d, text: 解答。, finishReason: null
requestId: 6ec84d07-e933-9c88-a53a-d1a00bfd734d, text: , finishReason: stop
requestId: 0ab0fabc-1d9b-94db-89bb-c8baccaea2ab, text: 你好！, finishReason: null
requestId: 0ab0fabc-1d9b-94db-89bb-c8baccaea2ab, text: 欢迎, finishReason: null
```


- 我们需要对每一部分的消息片段放到一个**阻塞队列**中，这样前端就可以轮询获取流式的数据，实现如下
- 观察调用AI接口返回的数据，有一个**finishReason**字段，当值为“stop”时，说明这是一次回复的结尾，也就是说我们也要告诉前端，**本次的轮询获取数据结束了**，于是我们显式设置一个response的isEnd属性。

```
public class MessageUtil {

    private static final BlockingQueue<Response> messageResponseQueue = new
LinkedBlockingQueue<>();
    public static BlockingQueue<Response> getQueue(){
        return messageResponseQueue;
    }
    public static void streamCall(String userMessage) throws NoApiKeyException,
InputRequiredException {
        ApplicationParam param = ApplicationParam.builder()
                // 若没有配置环境变量，可用百炼API Key将下行替换为：api_key="sk-
xxx"。但不建议在生产环境中直接将API Key硬编码到代码中，以减少API Key泄露风险。
                .apiKey("APIKEY")
                .appId("APPID")
                .prompt(userMessage)
                .incrementalOutput(true)
                .build();

        Application application = new Application();
        Flowable<ApplicationResult> result = application.streamCall(param);

        result.blockingForEach(data -> {
                // 记录是否是本次回复的最后一段数据
            boolean isEnd = Objects.equals(data.getOutput().getFinishReason(),
"stop");
            // 这里返回的是部分数据，我们放入队列中
```

```
            messageResponseQueue.offer(new Response(isEnd,
data.getOutput().getText()));
            System.out.printf("requestId: %s, text: %s, finishReason: %s\n",
                    data.getRequestId(), data.getOutput().getText(),
data.getOutput().getFinishReason());
        });
    }
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27

前端

script部分

- 这里我们主要做的是用户发消息，调用send请求，并开启长轮询longPolling

```
<script>
// api部分大家根据自己的前端框架自己封装即可，分别调用后端的两个controller
```

```
import { longPolling, send} from "@/api/evaluate/project";
export default {
  data() {
    return {
      messages: [], // 消息记录
      userInput: "你好", // 用户输入，默认一开始发一个“你好”
      pollingActive: false, // 是否正在长轮询
      isEnd: false, // 标记是否结束轮询
      currentAiMessageId: null, // 当前正在回复的 AI 消息的 ID
      userMsgData: {}, // 用户消息数据
    };
  },
  async mounted() {
    this.sendMessage();
  },
  methods: {
    sendMessage() {
      if (!this.userInput.trim()) return;

      // 添加用户消息
      this.messages.push({
        id: Date.now(),
        content: this.userInput,
        from: "user",
      });

      this.userMsgData.content = this.userInput;
      send(this.userMsgData);

      // 清空输入框
      this.userInput = "";

      // 添加 AI 回复占位
      const newAiMessage = {
        id: Date.now() + 1,
        content: "",
        from: "ai",
      };
      this.messages.push(newAiMessage);

      this.currentAiMessageId = newAiMessage.id;

      // 启动轮询
      if (this.isEnd || !this.pollingActive) {
```

```
        this.isEnd = false;
        this.pollingActive = true;
        this.polling();
      }
    },
    async polling() {
      try {
        const response = await longPolling();
        let newMessageContent = response.content.trim();

                // 通过消息id获取目前的AI输入位置
        const aiMessage = this.messages.find(
          (msg) => msg.id === this.currentAiMessageId
        );

        if (aiMessage) {
          aiMessage.content = `${aiMessage.content}${newMessageContent}`.trim();
        }

                // 如果是最后一段数据，则停止轮询
        if (response.end) {
          this.isEnd = true;
          this.pollingActive = false;
        } else if (this.pollingActive) {
          this.polling();
        }
      } catch (error) {
        console.error("长轮询失败:", error);
        if (this.pollingActive) {
          setTimeout(this.polling, 5000);
        }
      }
    },
  },
};
</script>
```

- 1
- 2
- 3
- 4
- 5
- 6

- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35
- 36
- 37
- 38
- 39
- 40
- 41
- 42
- 43
- 44
- 45
- 46
- 47
- 48
- 49
- 50

- 51
- 52
- 53
- 54
- 55
- 56
- 57
- 58
- 59
- 60
- 61
- 62
- 63
- 64
- 65
- 66
- 67
- 68
- 69
- 70
- 71
- 72
- 73
- 74
- 75
- 76
- 77
- 78
- 79
- 80
- 81
- 82
- 83

template和style部分

```
<template>
  <div class="app-container">
    <!-- 聊天界面 -->
    <div class="chat-container">
      <!-- 消息展示区域 -->
      <div class="chat-box">
        <div
          v-for="message in messages"
```

```
          :key="message.id"
          class="message"
          :class="message.from === 'user' ? 'user-message' : 'ai-message'"
        >
          <p>{{ message.content }}</p>
        </div>
      </div>
      <!-- 输入框与发送按钮 -->
      <div class="input-container">
        <el-input
          v-model="userInput"
          placeholder="请输入消息..."
          clearable
          @keyup.enter.native="sendMessage"
          class="chat-input"
        />
        <el-button type="primary" icon="el-icon-send" @click="sendMessage"
class="send-button">发送</el-button>
      </div>
    </div>
  </div>
</template>


<style scoped>
.app-container {
  display: flex;
  height: 90vh;
  background-color: #f3f4f6;
  font-family: "Arial", sans-serif;
}

/* 聊天容器 */
.chat-container {
  flex: 1; /* 右侧占比 */
  display: flex;
  flex-direction: column;
  border-left: 1px solid #ddd;
  background-color: #fff;
  overflow: hidden;
}

.chat-box {
  flex: 1;
```

```
  overflow-y: auto;
  padding: 20px;
  background-color: #fafafa;
  display: flex;
  flex-direction: column;
}

/* 通用消息样式 */
.message {
  margin: 10px 0;
  padding: 10px;
  max-width: 70%;
  word-wrap: break-word;
  border-radius: 8px;
}

/* 用户消息：右对齐 */
.user-message {
  align-self: flex-end;
  background-color: #e0f7fa;
  text-align: left;
}

/* AI 消息：左对齐 */
.ai-message {
  align-self: flex-start;
  background-color: #f1f1f1;
  text-align: left;
}

/* 输入框和发送按钮 */
.input-container {
  display: flex;
  padding: 10px;
  border-top: 1px solid #e0e0e0;
  background-color: #f9f9f9;
}

.chat-input {
  flex: 1;
  margin-right: 10px;
}

.send-button {
```

```
  flex-shrink: 0;
}

</style>
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35
- 36
- 37
- 38

- 39
- 40
- 41
- 42
- 43
- 44
- 45
- 46
- 47
- 48
- 49
- 50
- 51
- 52
- 53
- 54
- 55
- 56
- 57
- 58
- 59
- 60
- 61
- 62
- 63
- 64
- 65
- 66
- 67
- 68
- 69
- 70
- 71
- 72
- 73
- 74
- 75
- 76
- 77
- 78
- 79
- 80
- 81
- 82

- 83
- 84
- 85
- 86
- 87
- 88
- 89
- 90
- 91
- 92
- 93
- 94
- 95
- 96
- 97
- 98
- 99
- 100