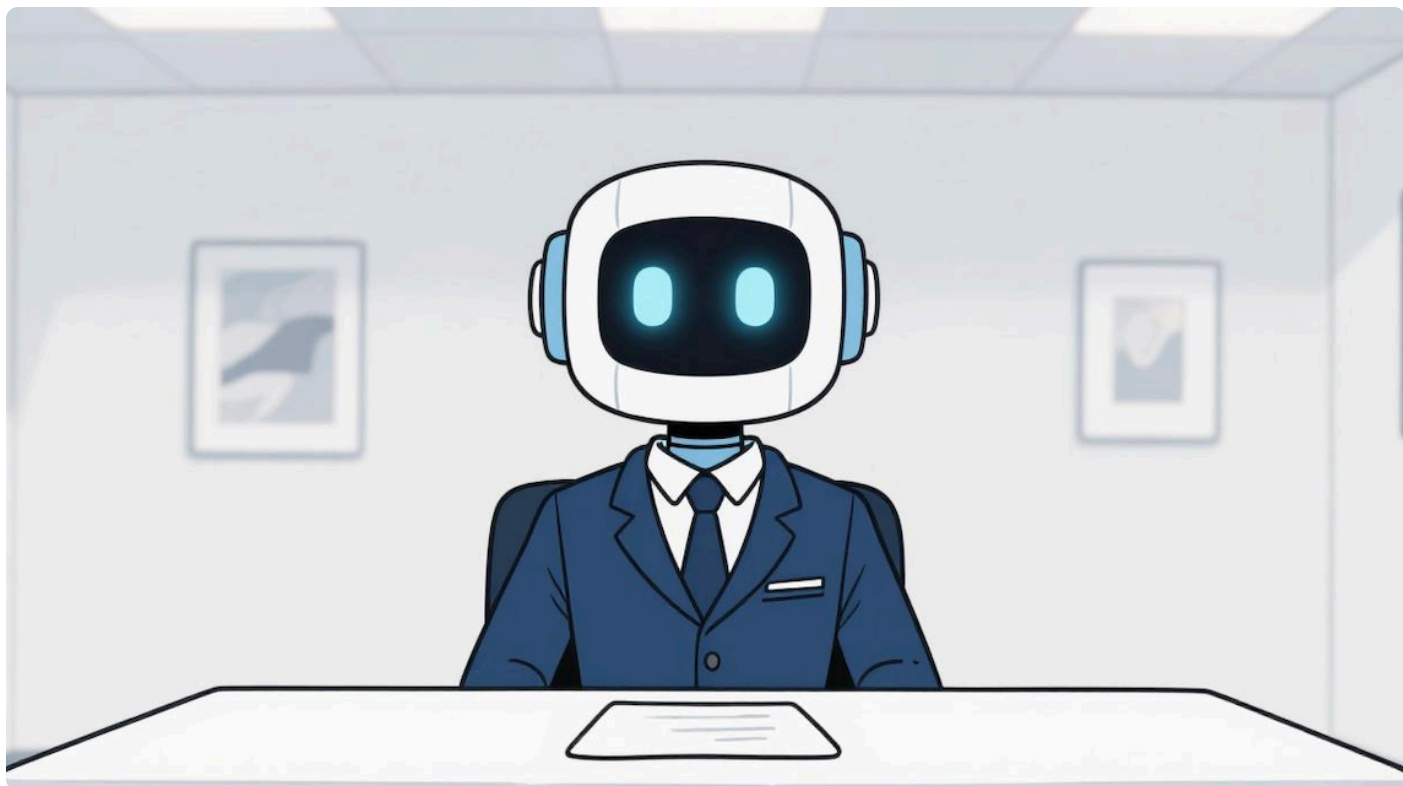


22 | 项目概览：AI前端面试官

月影 · 跟月影学前端智能体开发



你好，我是月影。

从这节课开始，我们进入一个新的单元。

上一个单元，我们从零到一开发了针对儿童科普的 AI 应用——波波熊学伴。它的核心 workflow，是**根据用户输入的话题，通过一系列的知识构建和内容组织，将它以特定用户群体所希望的形式表达出来**。这是一类非常典型的 AI 应用，通常学习类、工具类应用具备这样的特点。

在这一单元，我们将尝试另外一种类型的 AI 应用，这种 AI 应用的特点是**针对某类特定目的，以复杂的、结构化的对话流形式完成用户希望的互动，从而达成目标**。这是另外一种典型 AI 应用，通常像智能客服、智能导购类的应用具备这样的特点。

AI 前端面试官产品规划

接下来我们实现的产品是模拟面试官，因为我们比较熟悉自己的前端领域，所以我们就以前面试官为例来讲解。

梳理业务流程

在这里，我们先梳理前端面试官的主体流程，我相信很多同学都经历过面试，应该对这一套流程比较熟悉。

在这里我们以大厂的前端面试官一面流程为规范，通常一轮面试包括以下几个阶段：

1. 面试开场、自我介绍
2. 讨论项目经历
3. 讨论技术问题
4. 讨论代码 / 解算法题
5. 非技术面试阶段
6. 面试结尾

在面试开始时，面试官一般会让候选人做一下自我介绍，之后面试官会就简历和自我介绍中涉及到的项目经历深入探讨一些问题，从候选人过往经历中考察候选人的基础和发展潜力。

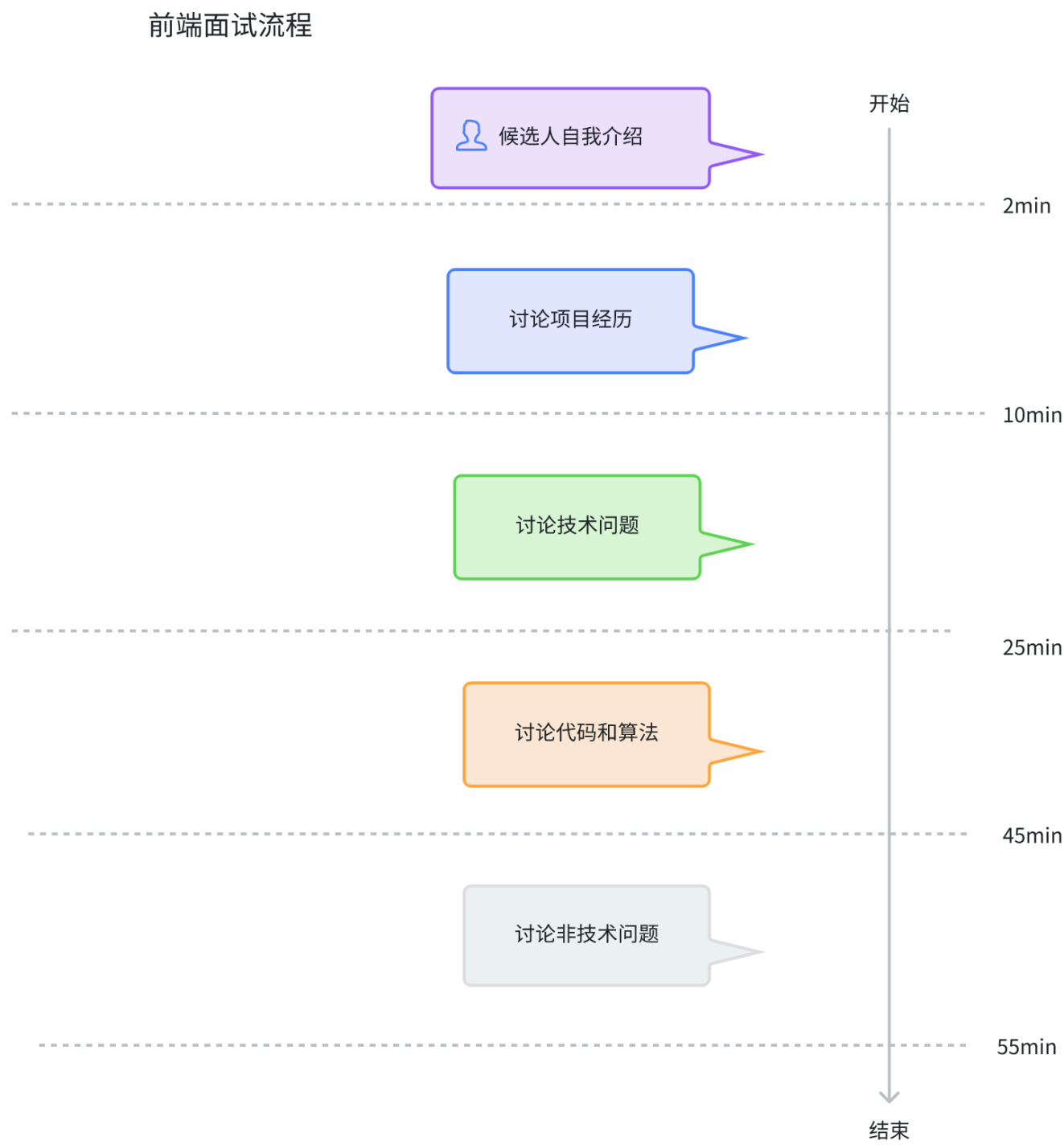
接下来随着讨论深入，面试会过渡到具体知识点，例如前端框架使用、事件机制、异步流程、CSS 运用等等一些前端知识点的考察。

在这些问题之后，面试自然过渡到讨论代码和解算法题的阶段，在这个阶段，面试官会和候选人讨论一些代码运行结果，或者出一道算法题让候选人给出结果。

最后，面试进入非技术面试阶段，此时面试官会问面试人一些沟通协调、团队合作、项目管理方面的经验和认知，从而考察面试人的软素质。

在这个环节结束后，面试官会让候选人提问，解答候选人一些感兴趣的问题，然后结束整个面试。整个面试过程一般持续 45 分钟到 1 个小时。

我们可以把整个面试流程用下面的图来概括表示。



所以我们要解决的主要问题，就是让 AI 面试官能够主导并把控整个对话流程和节奏，这个不是通过随便聊天可以精准控制的，从技术上来讲比较有挑战。

解决问题的两个选择方向

要攻克这个流程的问题，大体上有两个技术方向可选择。

其一是选择依赖大模型的**长上下文多轮对话能力**。一些优秀的大模型，在长上下文的多轮对话表现相当不错。这一块目前国外的大模型如 GPT-4o、Gemini 2.0 或 Claude 3.7，在多轮对话方面表现都相当不错，国内的模型则差强人意。

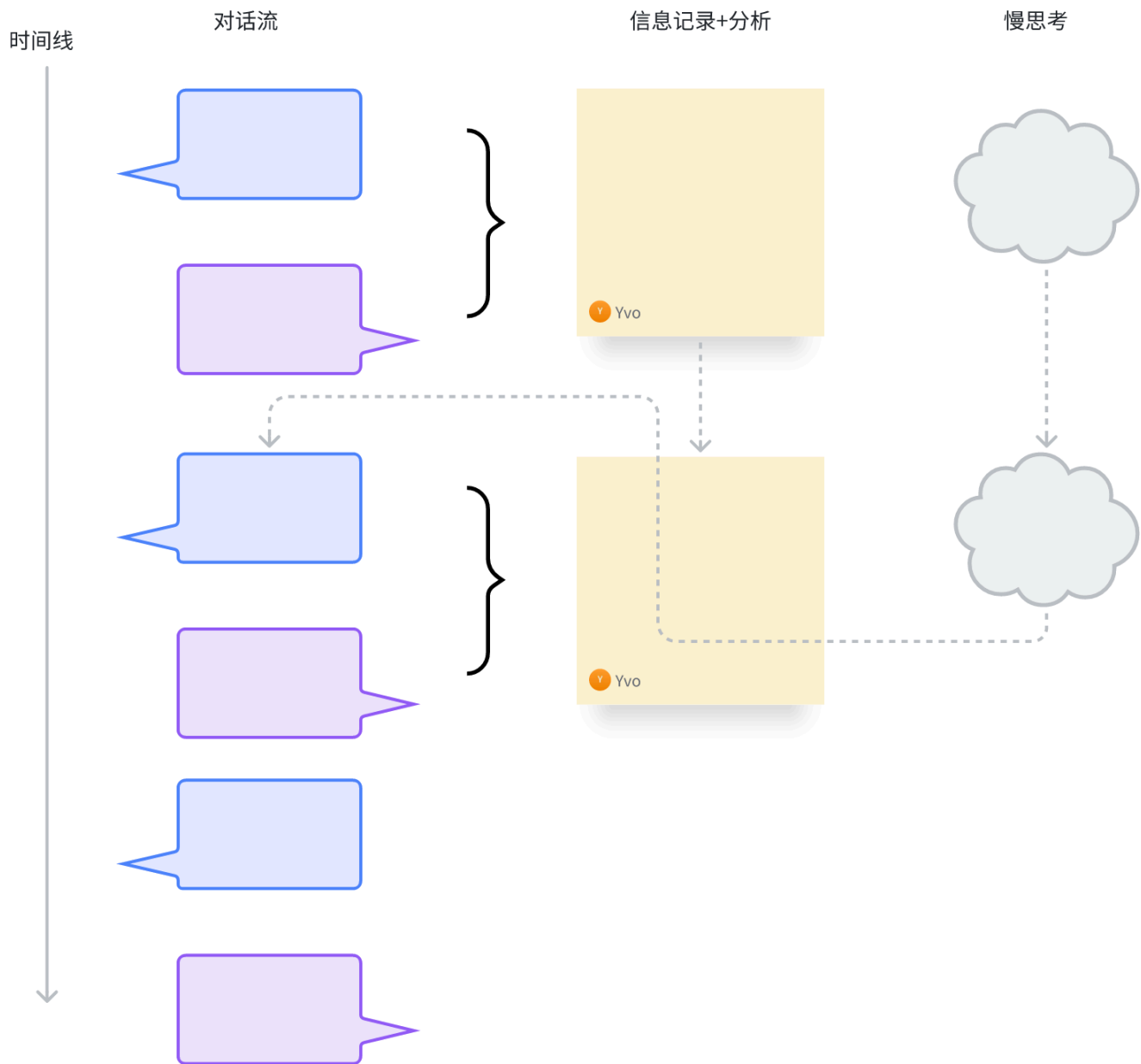
如果我们选择的大模型，在长上下文多轮对话能力上表现较好，那么理论上说，我们可能不用太多工程化的流程，只需要有时间节奏控制，将上面的六个阶段告知大模型，让大模型发挥就好，这样我们的核心工作流就非常简单了。

不过，理想是丰满的，现实是骨感的。因为面试对话有非常严格的流程和目的性，所以即使用了很强的模型，我们要精准把握面试结构，难度也是非常大的。

所以如果选择这个技术方向，根据我们的调研，目前基本上是无法达到令人非常满意的效果的，而且即使能勉强达到效果，后续的改进空间有限，极度依赖大模型本身的能力突破。即便未来真的能够突破，这种多轮对话随着轮次增加，token 的消耗也会大大增加，从成本上来说也并不划算，还很难后期优化。

因此我们只能考虑第二种选择方向，即不依赖于长上下文多轮对话，而是将流程拆解成**对话、信息分析、慢思考**三个部分，用 MoE（混合专家模型）的思路来解决这个问题。

具体做法是这样的，我们将多轮面试对话工作流，拆分成并行的三部分流程，如下图所示：



我们的面试主体流程有三个并行的**异步**工作流。

对话工作流

一个对话工作流负责实时对话，基本上是面试官和候选人一问一答。

记录分析工作流

记录分析 workflow 负责内容的收集和分析，例如在前面的对话中，面试官与候选人交流了什么，从候选人的回答中，面试官得出了什么样的判断，在这个过程中候选人有什么额外的需要关注的地方等等，这些东西都被记录下来。

慢思考 workflow

慢思考 workflow 会结合这些信息，分析出接下来后续面试过程中面试官需要重点关注的内容和采取的策略，将这些内容反馈给对话 workflow，指导对话 workflow 下一步的输出。

这就像是真正的一个有经验的面试官，一边和候选人聊，一边会记录一些沟通要点，同时脑子里思考接下来要如何和候选人进一步交流。只有这样，才能保证问到点子上，并最终给出对候选人客观、深刻的评价。


其他细节技术选型和项目搭建

解决了 workflow 的问题，我们再进一步梳理产品必要的一些技术细节和解决方案。

面试过程中，我们在讨论代码和算法阶段，会让候选人写代码，所以我们在 UI 层需要一个网页的代码编辑器，目前主流的方案是 CodeMirror 或 Monaco Editor，但我们主要支持的是前端的 JavaScript 语言，需求也比较简单，所以用 CodeMirror 更轻量和方便。而且我们的项目是 Vue3，所以我们直接用 vue-codemirror 就可以了。

接下来我们在 Trae 创建一个新的 Vue+TypeScript 项目 Get Offer AI。

然后我们配置一下 .local.env。


 复制代码

```
1 VITE_DEEPSEEK_API_KEY=sk-2621*****262b
2 VITE_DEEPSEEK_ENDPOINT=https://api.deepseek.com/chat/completions
3 VITE_DEEPSEEK_MODEL=deepseek-chat
```

这次我们大模型选择了 deepseek-v3。


与波波熊学伴项目相同，我们依然采用如下项目结构：

```
1 |— lib
2 |   |— prompt
3 |   |— service
4 |— src
5 |— server.ts
```

 复制代码


在项目目录中创建 `server.ts`，安装必要的依赖。

```
1 pnpm i jiti dotenv @bearbobo/ling express body-parser vue-codemirror
```

 复制代码


然后是 TypeScript 类型依赖。

```
1 pnpm i -D @types/express @types/body-parser
```

 复制代码


安装依赖后，我们创建 `server.ts` 的默认内容。

```
1 import * as dotenv from 'dotenv';
2 import express from 'express';
3 import { type ChatConfig, Ling } from "@bearbobo/ling";
4 import { pipeline } from 'node:stream/promises';
5
6 dotenv.config({
7   path: ['.env.local', '.env']
8 });
9
10 const apiKey: string = process.env.VITE_DEEPSEEK_API_KEY as string;
11 const endpoint: string = process.env.VITE_DEEPSEEK_ENDPOINT as string;
12 const modelName: string = process.env.VITE_DEEPSEEK_MODEL_NAME as string;
13
14 const app = express();
```

 复制代码


```
15 const port: number = 3000;
16
17 app.get('/', (req, res) => {
18     res.send('Hello, World!');
19 });
20
21 // 启动服务器
22 app.listen(port, () => {
23     console.log(`Server running on http://localhost:${port}`);
24 });
```

然后修改 package.json 的启动脚本配置。

 复制代码

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

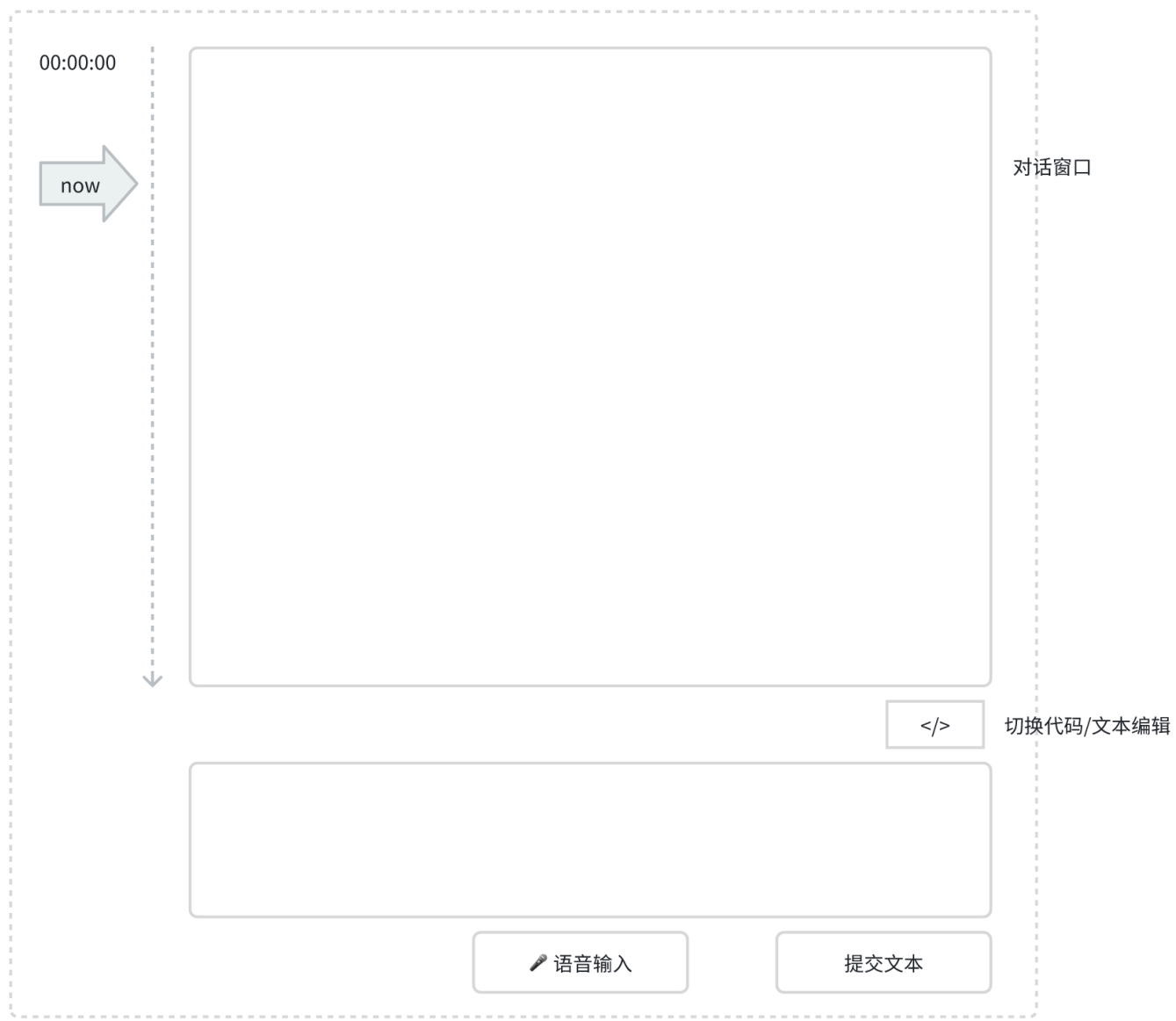
以及 vite.config.ts 的 server proxy 配置。

 复制代码

```
1  ...
2  export default defineConfig({
3    server: {
4      allowedHosts: true,
5      port: 4399,
6      proxy: {
7        '/api': {
8          target: 'http://localhost:3000',
9          secure: false,
10         rewrite: path => path.replace(/^\/api/, ''),
11       },
12     },
13   },
14   ...
```

这样我们就完成了前端面试官项目的 Server 准备工作。

接下来我们设计一个简单的 UI 主界面。



如上图所示，产品整个主体界面是这样的。左侧一个时间轴，是一个独立的 UI 组件，用来显示当前面试时间进度。右侧上部是聊天窗口，显示历史对话。下方是一个输入框，可以输入文本，也可以切换成代码输入，主要是在讨论算法阶段有可能要使用。然后底部有两个按钮，分别是语音输入切换按钮和文本提交按钮。

这样的话，我们整体界面就设计完成了。从第一句聊天对话被发送时面试将正式开始，直到最后面试官结束面试，我们主体流程都将在这个界面上完成。

接下来还有一些产品细节要考虑，比如面试结束后，面试结论以什么方式发送和呈现，这里我打算将它简化为面试结论卡片的形式，在面试结束后直接以发送到主界面对话流中。

另外，原本的产品还有管理职位、设置面试级别、难度，以及提交候选人简历等功能。在这里我们将这些非核心功能做一些简化，不提供 UI 交互，而是把它们抽取成配置，在后续课程中具体到相关细节时，我们再展开一一讨论。

要点总结

从这一节课开始，我们进入了一个新的单元，以一个典型的复杂流程多轮对话产品——AI 前端面试官威力，通过拆解和实现它的主题流程，带你通过实战学习并领会如何开发这一类型的 AI 应用产品。

通常情况下，解决复杂多轮对话问题，有两个技术突破方向，一是依赖于强大的大模型本身对长上下文多轮对话的支持，可能更轻松地松解问题。但是这里我推荐的更好的方法是通过 MoE（混合专家模型），将对话、信息整理分析和慢思考拆分开来，通过异步工作流的配合去解决这一问题，这么做能够更可控，也更低成本地解决问题，从而达到更好的效果。

接下来我们会通过几节课分别深入细节，一步步将产品逐步实现。

课后练习

在课程正式开始之前，你可能心里面有疑问：我们前面说的第一种方式，依赖于支持长上下文多轮对话的强大模型究竟可不可以？

你可以试着直接用 ChatGPT、Gemini 或者 Claude 扮演前端面试官，走一轮面试聊天流程，看看是否能够达到我们期望的面试节奏和效果。如果能达到，那是怎么做到的？如果不能达到，又是遇到了什么具体问题呢？你可以将你的研究分享到评论区。

AI智能总结

1. 介绍了AI前端面试官产品规划和前端面试官的主体流程，包括面试开场、自我介绍、讨论项目经历、讨论技术问题、讨论代码/解算法题、非技术面试阶段和面试结尾。
2. 提到了解决问题的两个选择方向，一是依赖大模型的长上下文多轮对话能力，二是将流程拆解成对话、信息分析、慢思考三个部分，用MoE（混合专家模型）的思路来解决问题。
3. 对话工作流、记录分析工作流和慢思考工作流是解决问题的具体做法，分别负责实时对话、内容的收集和分

- 析，以及结合信息分析出接下来后续面试过程中需要重点关注的内容和采取的策略。
- 4. 建议采用不依赖于长上下文多轮对话的解决方案，因为选择依赖大模型的长上下文多轮对话能力在实际应用中存在难度和成本问题。
 - 5. 提出了将多轮面试对话 workflow 拆分成并行的三部分流程的具体做法，以实现对话流程和节奏的精准控制，从而达到目标。
 - 6. 介绍了产品必要的一些技术细节和解决方案，包括网页的代码编辑器选择、项目搭建和配置，以及前端面试官项目的Server准备工作。
 - 7. 设计了产品的UI主界面，包括左侧时间轴、右侧聊天窗口、输入框和按钮，以及面试结束后的结论卡片形式呈现。
 - 8. 提到了解决复杂多轮对话问题的两个技术突破方向，以及推荐的更好的方法是通过MoE（混合专家模型）来解决问题，以实现更可控、更低成本地解决问题。
 - 9. 进入了一个新的单元，以一个典型的复杂流程多轮对话产品——AI前端面试官威力，通过拆解和实现它的主流程，带你通过实战学习并领会如何开发这一类型的AI应用产品。
 - 10. 提出了课后练习，鼓励尝试依赖于支持长上下文多轮对话的强大模型的方式，并分享研究结果到评论区。

© 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。