Legal-Chat 前端接口文档

更新日期：2023-06-02

介绍

LegalChat 是法狗狗基于大语言模型技术研发的可行法律咨询机器人，目前已经导入绝大部分全国性法律法规、司法解释，导入数百万条律师建议和判决案例。该咨询机器人具有理解用户意图、主动引导用户补充信息、解答用户法律问题等功能。接口使用 Websocket 协议传输，机器人回答问题时，会逐字流式返回咨询结果。由于该服务处于开发测试阶段，该接口的数据结构和用法可能随时变更，请您谅解。

示例代码为 TypeScript+Vue3 代码，其他语言可以参考该结构。该代码不是完整代码，无法直接运行。仅供您参考数据结构和对接方式。

建议框架

Web：TypeScript + Vue3 + Vite

安卓：Kotlin + Jetpack Compose 或其他有双向数据绑定功能的框架

IOS：Swift UI

桌面：Electron 等跨平台框架+Web 技术栈

注意事项

1. 一个问答一个 Websocket， 也就是说每次发起问题都需要新建一个 Websocket， 因为每次 Websocket 回答完成，后端都会把 Websocket 关闭
2. 一个话题的字数不能超过 120 字。
3. 如果上下文过长，后端会发送 type 为 close 的事件，也就是话题结束事件，需要用户重新咨询。

接口基本信息

token:

```
*****(请联系 FGG 商务获取)*****
```

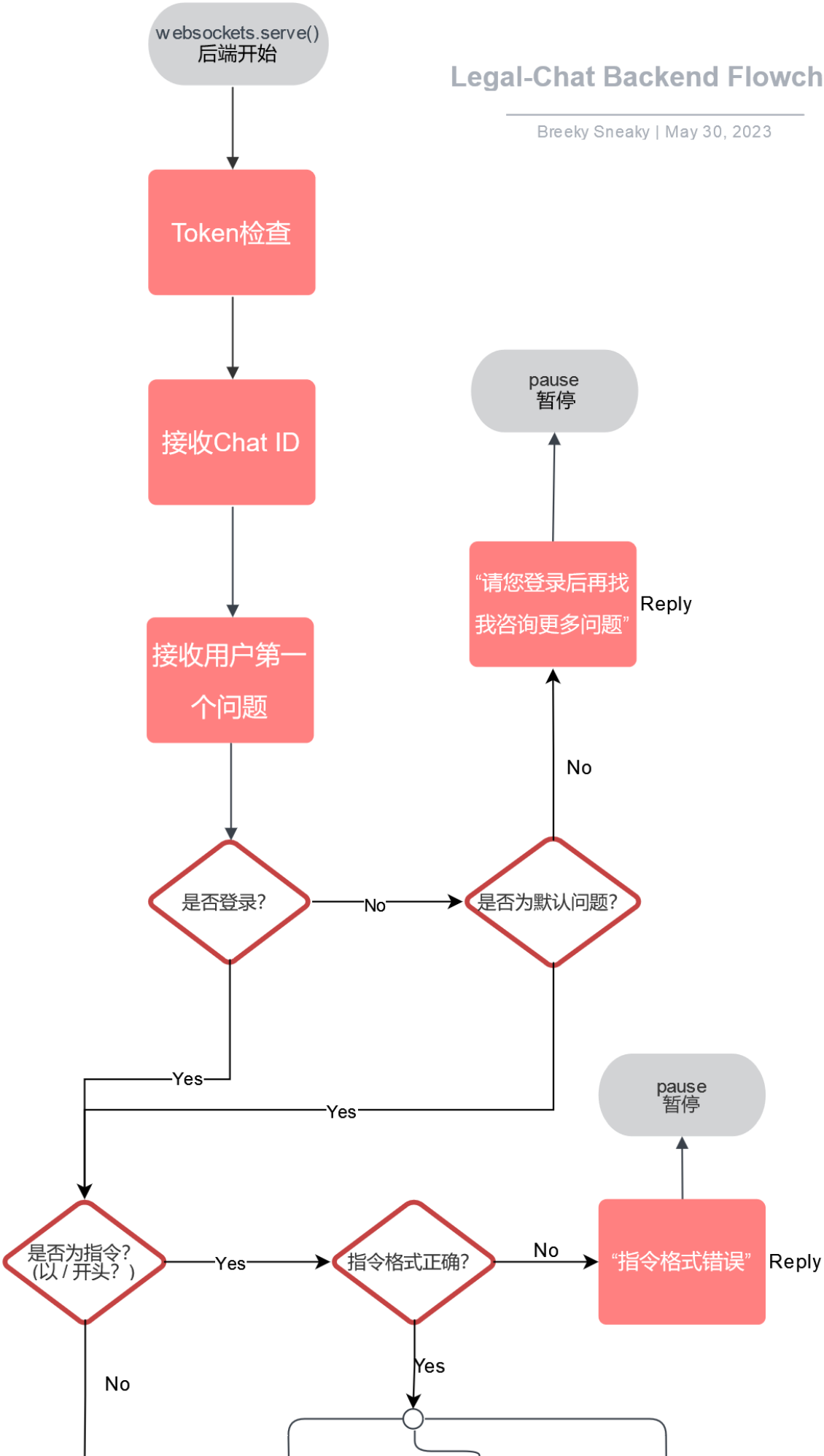
WsURL:

```
*****(请联系 FGG 商务获取)*****
```

后端服务流程图

Legal-Chat Backend Flowchart

Breeky Sneaky | May 30, 2023



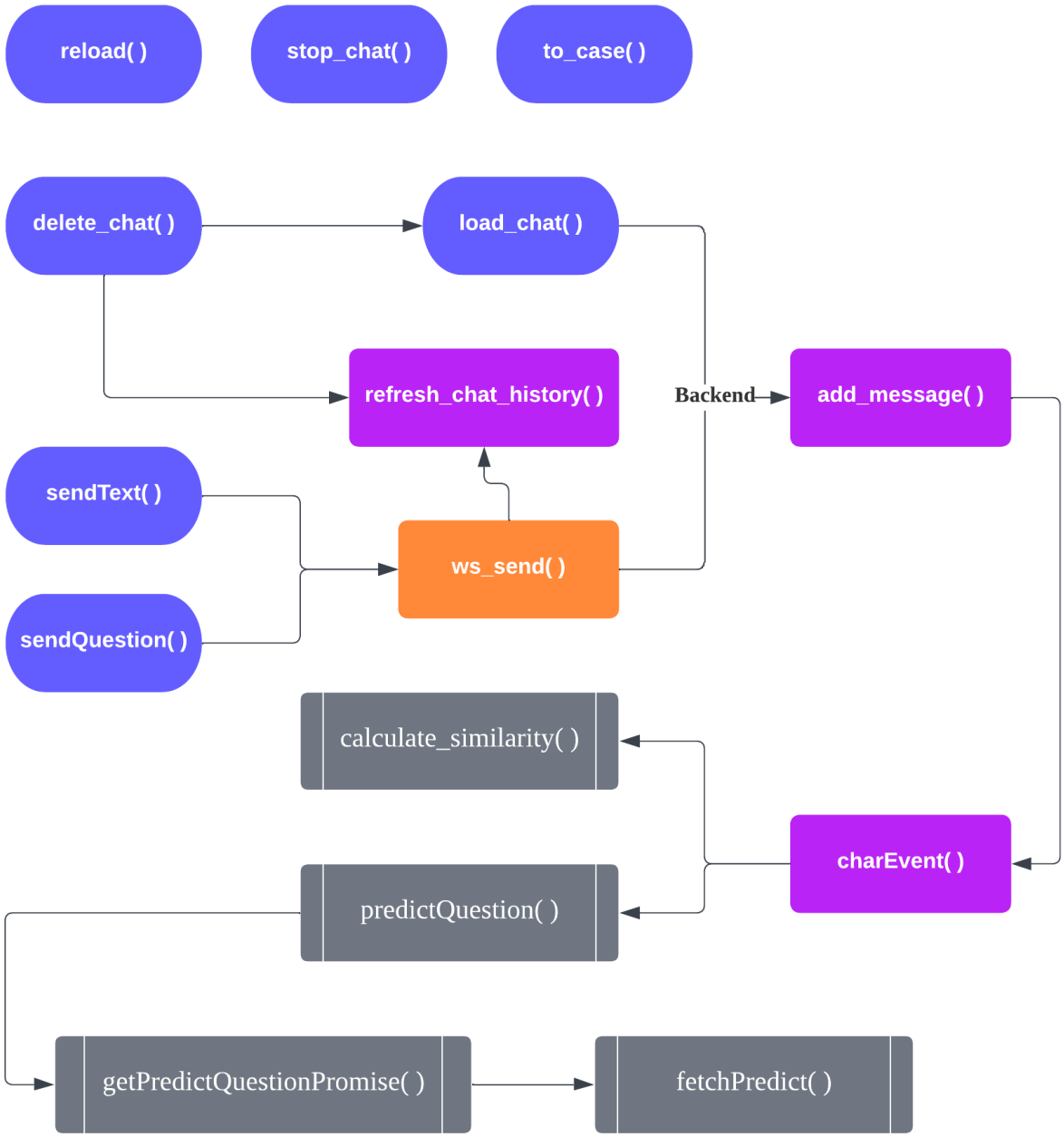
前端功能调用关系图

注：蓝色 - 核心业务功能，紫色 - 辅助业务功能，橙色 - 接口，灰色 - 问题预测/法律索引功能

在该文档搜索方法名可以看到该方法的具体实现代码

Frontend Calling Relational Diagram

Breeky Sneaky | June 1, 2023



接口说明

WebSocket 接口

请求地址：

*****(请联系 FGG 商务获取)*****

鉴权方式：

建立 WebSocket 连接时发送 user_id 作为首条消息

调试试用 Token：

*****(请联系 FGG 商务获取)*****

鉴权示例代码：

```
const ws = new WebSocket(target_url)
    ws.onopen = () => {
        loading.value = true
        stage_id = nanoid(10)
        let content_item = new Map([[text, []]])

        history.value.push(new MessageItem('user', content_item))
        recommend_list.value = []

        ws.send(localStorage.getItem('access_token') || '0')
        ws.send(chat_id.value)
        ws.send(text)
    }
```

发送数据：

1. Token

例：

eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6Ik9aSDJFNjlrZUROYzluSEdfemRwaGxqSjJ1b2tHaW5SRml0ajdxTldOS3http://cifQ.eyJzdWIiOiI2NDU0NTVlYjlkNThhY2QzNzZjNGVjNWIiLCJhdWQiOiI2NDQwOTM0MzI0ZTAyNGEwZjI3YjFmZjUiLCJzY29wZSI6Im9wZW5pZCBwcm9maWxlIGVtYWlsIHBob25lIGFkZHJlc3MiLCJpYXQiOjE2ODUzNDQ5MjQsImV4cCI6MTY4NjU1NDUyNCwianRpIjoiZWJlaDUxM21TQWhWajVTMHhtemtnZnJxWDVmMGppTVRoc21RRkNqYldhMSIsImlzcyI6Imh0dHBzOi8vbGVnYWwtY2hhdC5hdXRoaW5nLmNuL29pZGMifQ.LHa4oBLl4oX-qPANmfSJpVgd3Kj_xUi8cSu0GN8Q-3J_3_B05OSsD8xzwiAYF-kZQ3C_iZBvpJWltk_aerhS7FGsi6Kp_DO5daqvne7Af6uLO6T9JWZdLRR4QVZZ5CkVk84AokI2ifQJcrum6BDO8CHPLSHsFff9391GMTBScjemdPnPQyUnvBiHZPfTZsWSiTW_SJBl2R78gPf5jfoSROuIqrfLxZo23s4l9j3X2gBfLbM6g3bAecP2ZWEztnu233p6mPvpXBe0oDhFwu2ywz21U_SdbUSztqX13W_DOWcP-nZ3Zt2tnz6vhVqwB3YfmNaVMSl-rgsZMBNU3ycqNg

2. chat_id（10 位以上的 nanoid）

例：E3ewWLL3jN

3. 直接发送用户问题

例：我租的房子协议到期了，但房东不退押金，请问我该怎么办？

示例代码：

```
    ws.onmessage = (data: any) => {
            let event: any = JSON.parse(data.data)
            add_message(event)  // 具体详见下一小节
        }
        ws.onclose = () => {
            refresh_chat_history()  // 具体详见下一小节
            loading.value = false
        }
        ws.onerror = (err) => {
            console.log(err)
            message.error('服务器连接失败')
            stage_id = nanoid(10)
            if (!is_recommend) input.value = text
            loading.value = false
        }
        return ws
```

TypeScript 实现参考

基本工具

全局常量

```
const history: Ref<Array<MessageItem>> = ref([]) // 聊天记录
const input = ref('') // 输入框
const stream = ref('') // 流式回复
const process_current = ref(0) // 法典阅读进度
const process_total = ref(0) // 法典长度
const recommend_list: Ref<Array<string>> = ref([]) // 推荐回复列表
const source: Ref = ref([]) // 法典原文
const cases: Ref = ref([]) // 案例列表
const is_chat_session_closed = ref(false) // 是否关闭
const loading = ref(false) // 是否加载中
const is_mobile = ref(false) // 是否移动端
const legal_chat_history: Ref<Array<any>> = ref([]) // 其他法律会话列表

const isDelHistoryScroll = ref(false) // 是否删除历史记录后滚动
const activeMode = ref<string>('法律 AI') // 当前模式
const pageLoading = ref(false)

const wsUrlMap: { [index: string]: string } = {
    法律 AI: 'wss://service-1dbgn4h6-1254426977.hk.apigw.tencentcs.com/release' + '/legal-chat'
} // WebSocket URL 表
```

全局变量

```
let last_ask = '' // 上一次提问
let stage_id = '' // 当前阶段 id,用于丢弃过期请求
let ws: WebSocket | null = null
```

推荐数据结构

```
type ContentItem = Map<string, Array<any>>

class MessageItem {
    role: string
    content: ContentItem

    constructor(role: string, content: ContentItem) {
        this.role = role
        this.content = content
    }
}
```

随机生成 chat_id

注：chat_id 只有在对话重置时才被更改

```
const lawChatId = nanoid(10) // 当前法律会话 id
const chat_id = ref(lawChatId) // 对话 id
```

Base URL

```
const BASE_URL_HK = 'https://service-1dbgn4h6-1254426977.hk.apigw.tencentcs.com/release'
const BASE_URL_GZ = 'https://service-ci0s5pf0-1254426977.gz.apigw.tencentcs.com/release'
```

核心业务功能

核心业务需与页面组件进行整合

发送预设问题

输入：问题字符串

输出：None

```
function sendText() {
    if (loading.value) return
    if (is_chat_session_closed.value) return
    if (!input.value) return message.warning('问题不能为空')
    if (input.value.length > 120) {
        message.error('输入长度不能超过 120')
        return
    }

    ws = ws_send(input.value, wsUrlMap[activeMode.value])
    last_ask = input.value
    stream.value = ''
    stage_id = nanoid(10)
    setTimeout(() => {
        input.value = ''
    }, 0)
}
```

打开判决案例

输入：cases(案例列表)里某一 case 的 id　　例：case_item.id

```
function to_case(id: string) {
    window.open(`/case/?id=${id}`)
}
```

加载对话

输入：id - chat_id

type - mode　　例：　‘法律 AI’

重新加载对话

```
function reload() {
    if (loading.value) return message.warning('正在回复中，请耐心等待...')
    location.reload()
}
```

删除对话

```
async function delete_chat(delete_chat_id: string) {
    let res = await fetch(`${BASE_URL_HK}/chat_history?id=${delete_chat_id}`, {
        method: 'DELETE',
        headers: { Authorization: localStorage.getItem('access_token') || '' },
    })
    let data: any = await res.json()
    if (data['message'] == 'success') {
        message.success('删除成功')
        isDelHistoryScroll.value = false
        await refresh_chat_history()

        const other_chats = legal_chat_history.value[0].id

        if (chat_id.value == delete_chat_id) {
            isDelHistoryScroll.value = true
            await load_chat(other_chats, activeMode.value)
        }
    } else message.error('删除失败')}
```

关闭对话

```
function stop_chat() {
  if (!loading.value) return
    ws?.close()
    let contentItem = new Map([[stream.value, []]])
    history.value.push(new MessageItem('assistant', contentItem))
    stream.value = ''
    loading.value = false
}
```

辅助业务功能

辅助核心功能

刷新会话记录

注：更新对话历史

```
async function refresh_chat_history() {
  let res = await fetch(`${BASE_URL_HK}/chat_history`, {
        method: 'GET',
        headers: { Authorization:
localStorage.getItem('access_token') || '' },
    })
    let data = await res.json()
    data = data['data']
    legal_chat_history.value = []
    //mode
    data?.forEach((item: any) => {
        legal_chat_history.value.push(item)
    })
}
```

后端输出结果处理

输入：event - 由后端返回的事件 object

```
async function add_message(event: any) {
    switch (event.type) {
        case 'char':
            charEvent(event)
            break
    case 'text':
            // mode
            if (event.role == 'assistant' && activeMode.value ===
'法律AI') {
                let hash_result =
sha256(event['content'].replaceAll('\n', ''))
                let res = await
fetch(`${BASE_URL_HK}/answer_source_cache?query=${hash_result}`)
                let obj = await res.json()
                let data: ContentItem = new
Map(Object.entries(obj))
                if (data.has('message')) {
                    let content_item = new
Map([[event['content'], []]])
                    history.value.push(new
MessageItem(event['role'], content_item))
                } else history.value.push(new
MessageItem(event['role'], data))
            } else {
                let content_item = new Map([[event['content'],
[]]])
                history.value.push(new MessageItem(event['role'],
content_item))
            }
            break
        case 'process':
            process_current.value = event.current
            process_total.value = event.total
            break
        case 'source':  // 可用可不用
            // mode
            if (event.data == 'clear') {
                source.value = []
                cases.value = []
            } else {
                source.value.push(event.data)
                if (event.data.类型 == '判决案例')
cases.value.push(event.data)
```

字符处理

注：对后端返回的内容文本进行处理

```
function charEvent(event: any) {
    process_total.value = 0
    if (event.content == '[结束]') {
        stage_id = nanoid(10)
        let temp_list = stream.value
            .replaceAll('\n\n', '\n\n|勷|')
            .replaceAll('。', '。|勷|')
            .replaceAll('? ', '? |勷|')
            .replaceAll('|勷|\n\n|勷|', '\n\n|勷|')
            .split('|勷|')
        let listA = []
        for (let item of temp_list) if (item.length > 0)
listA.push(item)
        // mode
        // 历史记录第一句不是问候语 并且用户问了问题
        if ((history.value[0] as MessageItem)?.role !==
'assistant') {
            predictQuestion({ role: 'assistant', content:
stream.value }, stage_id)
        }
        calculate_similarity(listA, source.value, stage_id)
    } else if (event.content == '[换行]') {
        let contentItem = new Map([[stream.value, []]])
        history.value.push(new MessageItem('assistant',
contentItem))
        stream.value = ''
    } else stream.value += event.content
}
```

问题预测/法律索引功能

对于生成文本的再处理

相似度计算

注：寻找与回答内容相关的案例/法律文件，并提供索引

输入：listA - 原文按照逗号句号切好的片段集合

listB - 数据库知识

curr_stage_id - 处理阶段 id

输出：ContentItem - 关联原文切段和数据库知识的 map

```
function calculate_similarity(
    listA: Array<string>,
    listB: Array<string>,
    current_stage_id: string
) {
    if (is_chat_session_closed.value) return
    const temp = history.value
    let message_map: ContentItem = new Map()
    for (let item in listA) message_map.set(listA[item], [])
    stream.value = ''
    history.value = temp.concat([new MessageItem('assistant',
message_map)])
    if (listB.length != 0 && !is_mobile.value) {
        fetch(`${BASE_URL_GZ}/similarity`, {
            method: 'POST',
            body: JSON.stringify({ listA: listA, listB: listB }),
        })
            .then((res) => {
                return res.json()
            })
            .then((data: any) => {
                if (stage_id != current_stage_id) return
                let message_map: ContentItem = new
Map(Object.entries(data))
                if (message_map.has('errorCode')) return
                history.value = temp.concat([new
MessageItem('assistant', message_map)])
            })
    }
}
```

问题预测

输入：last_message - dict 例：{role: 'assistant', content: 'Lorem Ipsum' }

```
function predictQuestion(last_message: any, current_stage_id:
string) {
    if (is_chat_session_closed.value) return
    recommend_list.value = []
    Promise.allSettled(
        getPredictQuestionPromise(last_message,
current_stage_id).map((item) =>
            item.func(item.args)
        )
    )
}

function getPredictQuestionPromise(last_message: any,
current_stage_id: string) {
    return new Array(3).fill(0).map(() => ({
        func: fetchPredict,
        args: { last_message, current_stage_id },
    }))
}
```

提取问题预测

```
async function fetchPredict({ last_message, current_stage_id }:
any) {
    let mode = 'law'
    fetch(`${BASE_URL_HK}/predict`, {
        method: 'POST',
        body: JSON.stringify({
            history: last_message,
            mode: mode,
        }),
    })
        .then((res) => {
            return res.json()
        })
        .then((res: any) => {
            if (stage_id == current_stage_id && res.data)
recommend_list.value.push(res.data)
        })
}
```