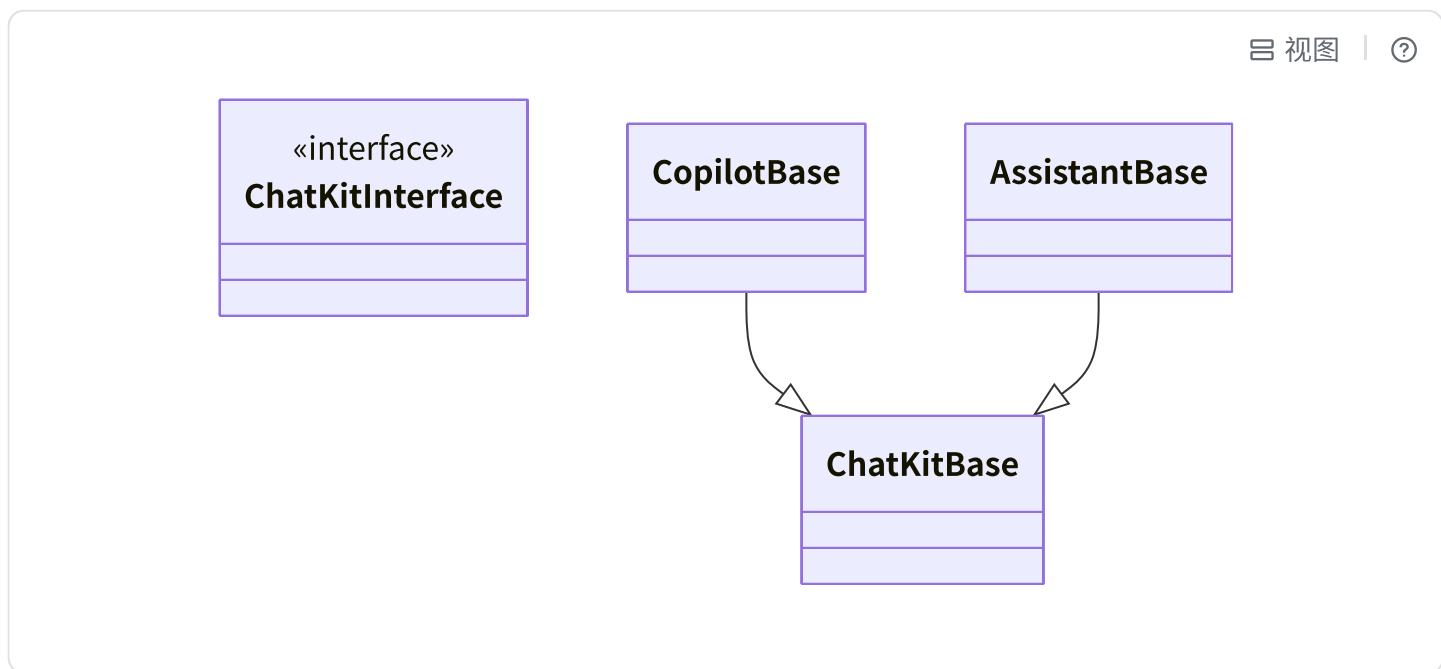


# ChatKit

## 第一部分 抽象接口和类设计

### 一、类关系图



### 二、抽象接口

#### 2.1 ChatKitInterface

该接口定义了 ChatKit 的一些抽象方法。

属性名	类型	说明
baseUrl	string	API 接口访问的根路径，默认为： /

##### 2.1.1 getOnboardingInfo(): OnboardingInfo

获取开场白和预置问题。该方法需要由子类继承并重写，以适配扣子、Dify 等 LLMOps 平台的接口。  
返回开场白信息结构体。

注意：该方法是一个无状态无副作用的函数，不允许修改 state。

## 2.1.2 `generateConversation(): string`

新建会话。该方法需要由子类继承并重写，以适配扣子、Dify 等 LLMOps 平台的接口。成功返回会话 ID。

注意：该方法是一个无状态无副作用的函数，不允许修改 state。

## 2.1.3 `sendMessage(text: string, ctx?: ApplicationContext, conversationID?: string): Promise<ChatMessage>`

向后端发送消息。该方法需要由子类继承并重写，以适配扣子、Dify 等 LLMOps 平台的接口。发送成功后，返回发送的消息结构。

注意：该方法是一个无状态无副作用的函数，不允许修改 state。

参数名	参数类型	说明
<code>text</code>	<code>string</code>	发送给后端的用户输入的文本。
<code>ctx</code>	<code>ApplicationConte xt</code>	随用户输入文本一起发送的应用上下文。
<code>conversationID</code>	<code>string</code>	发送的对话消息所属的会话 ID。

## 2.1.4 `terminateConversation(conversationID: string): Promise<void>`

终止对话。该方法需要由子类实现。

参数名	参数类型	说明
<code>conversationID</code>	<code>string</code>	当前正在接收 AI 助手消息的会话 ID。

## 2.1.5 `reduceAssistantMessage(eventMessage: T, prev: K): K`

将 API 接口返回的 EventStream 增量解析成完整的 AssistantMessage 对象。返回增量更新后的 AssistantMessage 对象。

注意：

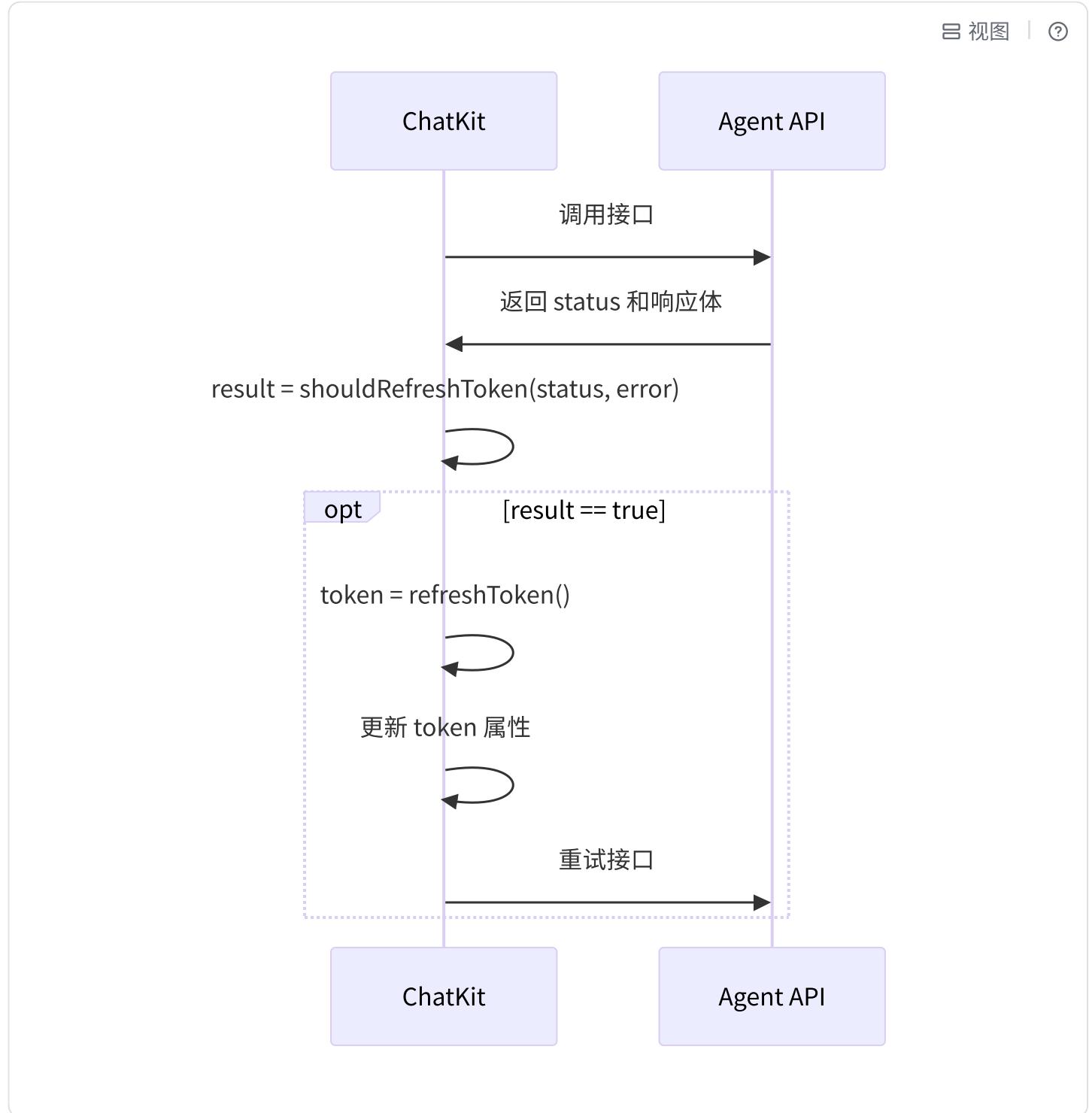
1. 该方法需要由子类实现。
2. 该方法在闭包中实现，解析完整个 EventStream 之后即丢弃。
3. 该方法是一个无状态无副作用的函数，不允许修改 state。

参数名	参数类型	说明

eventMessage	T	接收到的一条 Event Message
prev	K	上一次增量更新后的 AssistantMessage 对象

### 2.1.6 shouldRefreshToken(status: number, error: object): boolean

当发生异常时检查是否需要刷新 token。返回 `true` 表示需要刷新 token，返回 `false` 表示无需刷新 token。该方法需要由子类继承并重写，以适配扣子、Dify 等 LLMOps 平台的接口。



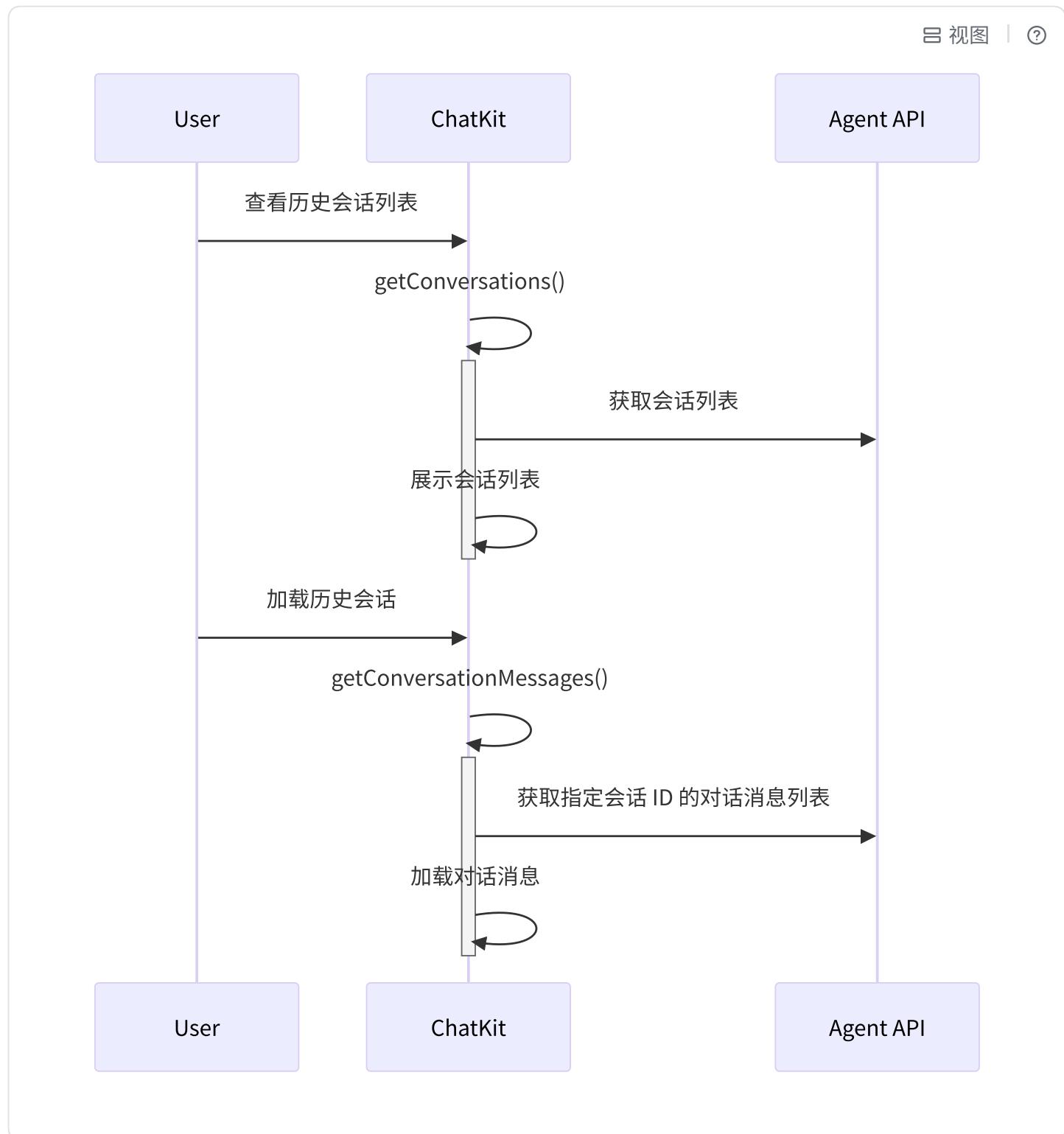
注意：重试接口后再次检查响应，如果仍然提示 token 失效，则放弃重新获取 token。

### 2.1.7 getConversations(): Array<ConversationHistory>

获取会话列表。

### 2.1.8 `getConversationMessages(conversationId): Array<ChatMessage>`

获取指定会话 ID 的对话消息列表。如果对话消息是 AI 助手消息，则需要调用 `reduceAssistantMessage()` 解析消息。



### 2.1.9 `deleteConversation(conversationID: string): void`

删除指定 ID 的会话。

## 三、类

### 3.1 ChatKitBase

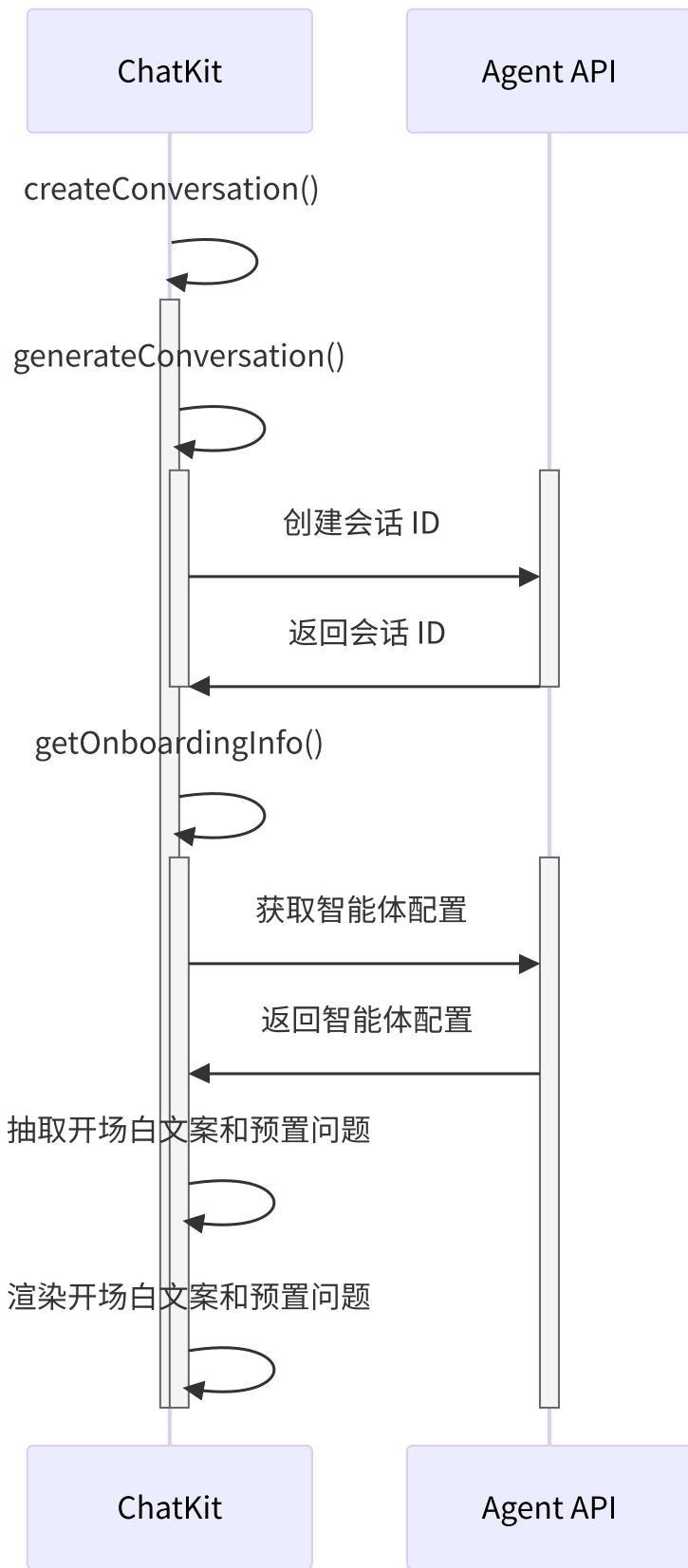
ChatKitBase 是 AI 对话组件的核心类。该类是一个 React 组件，包含标准的业务逻辑，但不包含交互界面和交互逻辑。

注意：开发者不能够直接挂载 `ChatKitBase` 到 Web 应用，而是需要创建一个子类继承 `ChatKitBase` 并实现 `ChatKitInterface` 中定义的方法。

属性名	类型	说明
<code>conversationID</code>	<code>string</code>	会话 ID，每次新建会话时由后端返回新的会话唯一标识。 在发送对话消息时，会将 <code>conversationID</code> 作为参数传入 <code>sendMessage()</code> 方法。
<code>messages</code>	<code>Array&lt;ChatMessage&gt;</code>	消息列表。
<code>textInput</code>	<code>string</code>	用户输入的文本。
<code>applicationContext</code>	<code>ApplicationContext</code>	和用户输入文本相关的上下文。
<code>defaultApplicationContext</code>	<code>ApplicationContext</code>	当没有指定 <code>applicationContext</code> 时的默认应用上下文。
<code>token</code>	<code>string</code>	调用接口时携带的令牌，放置到请求头： <code>Authorization: Bearer {token}</code>
<code>refreshToken</code>	<code>() -&gt; Promise&lt;string&gt;</code>	刷新 token 的方法，由集成方传入。

#### 3.1.1 `public createConversation(): void`

创建新的会话。`createConversation()` 方法内部会调用子类实现的 `generateConversation()` 方法。

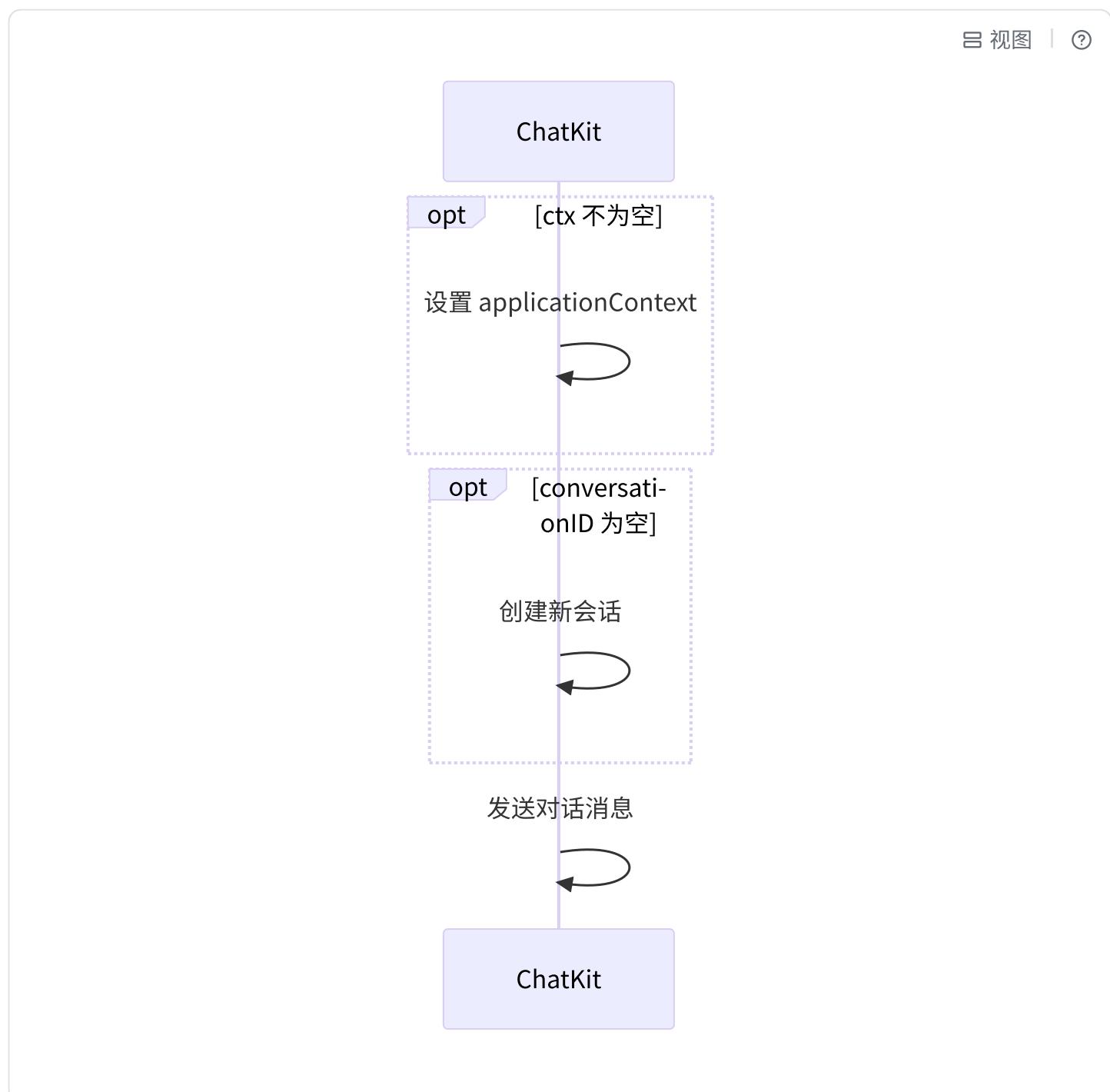


```
3.1.2 public send(text: string, ctx?: ApplicationContext,  
conversationID?: string): Promise<void>
```

发送消息。该方法是暴露给集成方进行调用的接口，`send()` 方法内部会调用子类实现的 `sendMessage()` 方法。

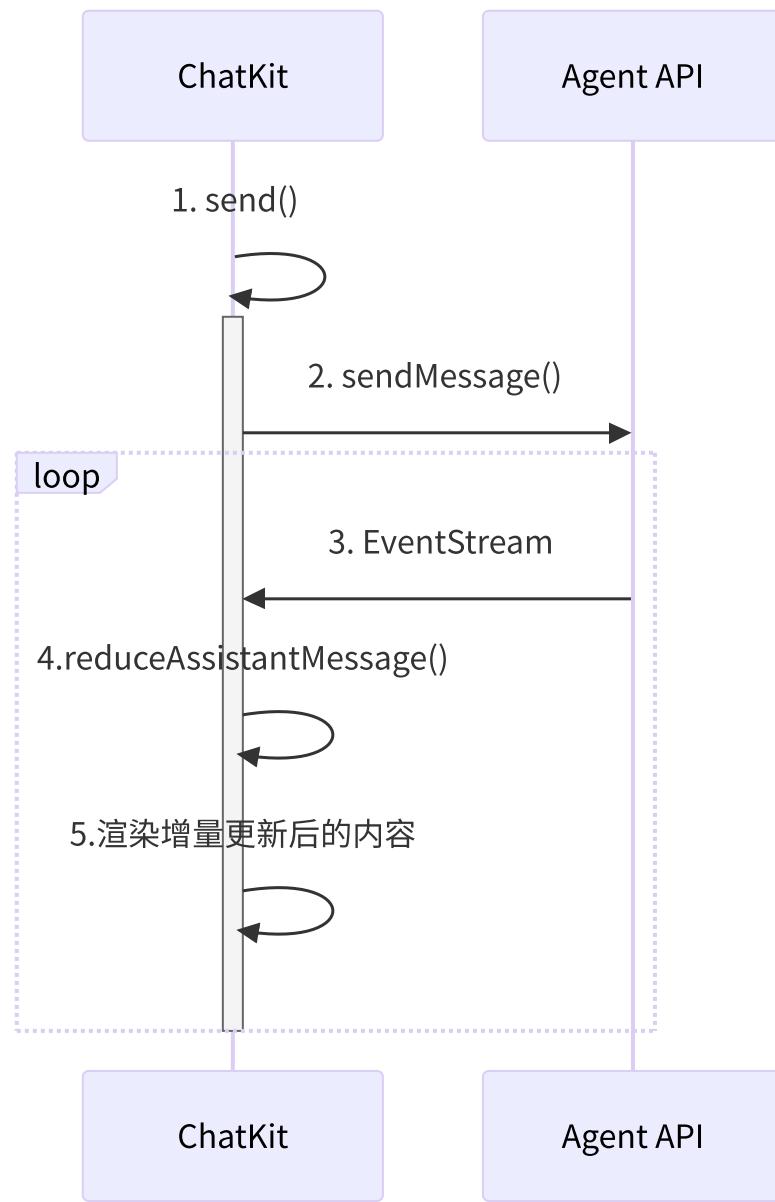
参数名	参数类型	说明
<code>text</code>	<code>string</code>	发送给后端的用户输入的文本。
<code>ctx</code>	<code>ApplicationContentxt</code>	随用户输入文本一起发送的应用上下文。
<code>conversationID</code>	<code>string</code>	发送的对话消息所属的会话 ID。

下图是调用 `send()` 方法时对应用上下文和会话的处理：



下图是 ChatKit 从 Agent API 接收处理 EventStream 数据流的流程：

视图 | ?



ChatKit 向 Agent 发起对话的消息处理流程：

1. ChatKit 调用 `send()` 方法发起处理流程。
2. `send()` 方法内部调用 `sendMessage()` 向 Agent API 服务发起对话请求（SSE）。
3. Agent API 服务持续输出 EventStream。
4. 调用 `reduceAssistantMessage()`，解析 Event Message 并增量更新到 `AssisatntMessage` 对象。
5. 从 `AssisatntMessage` 中提取需要渲染到界面的元素。这一步由子类实现抽取逻辑，再调用 `ChatKitBase` 提供的渲染方法。

```
3.1.3 public injectApplicationContext(ctx: ApplicationContext):  
void
```

向 ChatKit 注入应用上下文。

参数名	参数类型	说明
ctx	ApplicationContext	要注入的应用上下文

```
3.1.4 private clearConversation(): void
```

清除会话中的对话消息及会话ID。

```
3.1.5 private removeApplicationContext(): void
```

移除注入的应用上下文。

```
3.1.6 protected appendTextBlock(text: string): void
```

使用 Markdown 渲染 AI 助手返回的文本内容。该方法由子类调用。

参数名	参数类型	说明
text	string	要渲染到界面的文本，每次都传完整的文本。

```
3.1.7 protected appendWebSearchBlock(query: WebSearchQuery): void
```

渲染 AI 助手执行 Web 搜索的执行详情。该方法由子类调用。

参数名	参数类型	说明
query	WebSearchQuery	Web 搜索的执行详情

```
3.1.8 protected appendJson2PlotBlock(chartData: ChartDataSchema):  
void
```

参数名	参数类型	说明
chartData	ChartDataSchema	图标数据详情

渲染AI助手执行Json2plot生成图表数据，该方法由子类调用

### 3.1.9 protected appendExecuteCode(query: ExecuteCodeResult): void

渲染 AI 助手执行代码工具的详情。该方法由子类调用。

参数名	参数类型	说明
query	ExecuteCodeResult	代码执行的输入输出结果

## 3.2 CopilotBase

右侧跟随的 AI 助手，为应用提供辅助对话。该类继承自 ChatKitBase。

### 3.2.1 render(): React.ReactNode

实现抽象接口 `ReactComponent.render()` 方法。

## 3.3 AssistantBase

作为主交互入口，是应用的主体。该类继承自 ChatKitBase。

### 3.3.1 render(): React.ReactNode

实现抽象接口 `ReactComponent.render()` 方法。

## 四、结构体

### 4.1 ChatMessage

展示在消息区消息列表中的一条消息。

属性名	类型	说明
messageId	string	一条消息的 ID。
role	Role	发送该消息的角色。
content	Array<TextBlock   MarkdownBlock   WebSearchBlock>	该条消息的内容。一条消息可以由许多不同类型的消息块组成。

### 4.2 ConversationHistory

一条历史会话记录。

属性名	类型	说明

conversationID	string	会话 ID
title	string	会话标题
created_at	timestamp	会话创建时间
updated_at	timestamp	会话最后更新时间

## 4.3 ContentBlock<T, K>

消息块基类

属性名	类型	说明
type	T	消息块的类型。不同类型的消息块使用不同的组件进行渲染。
content	K	消息块的内容。

## 4.4 TextBlock

文本类型的消息块。

属性名	类型	说明
type	BlockType.Text	消息块的类型。
content	string	消息块的内容。

## 4.4 MarkdownBlock

Markdown 类型的消息块。

属性名	类型	说明
type	BlockType.Markdown	消息块的类型。
content	string	消息块的内容。

## 4.5 WebSearchBlock

Web 搜索类型的消息块。

属性名	类型	说明

type	BlockType.WebSearch	消息块的类型。
content	WebSearchQuery	消息块的内容。

## 4.6 WebSearchQuery

调用 Web 搜索的详情。

属性名	类型	说明
input	string	预置问题
results	Array<WebSearchResult>	Web 搜索结果集合

## 4.7 WebSearchResult

单条 Web 搜索的结果

属性名	类型	说明
content	string	搜索结果的内容摘要
icon	string	搜索结果的来源网站图标 URL
link	string	搜索结果的来源地址
media	string	搜索结果的来源网站名称
title	string	搜索结果的来源文章标题

## 4.8 OnboardingInfo

开场白信息，包含开场白文案和预置问题。

属性名	类型	说明
prologue	string	开场白文案
predefinedQuestions	Array<string>	预置问题

## 4.9 ApplicationContext

与用户输入的文本相关的应用上下文。

属性名	类型	说明
title	string	显示在输入框上方的应用上下文标题。
data	any	该应用上下文实际包含的数据。

## 4.10 Role

属性名	类型	说明
name	string	角色的名称： <ul style="list-style-type: none"><li>如果 type 是 Assistant，则名称为“AI 助手”</li><li>如果 type 是 User，则名称为用户的昵称/显示名</li></ul>
type	RoleType	发送该消息的角色
avatar	string	角色的头像，可以是 URL、Base64 或 SVG。

## 4.11 EventStreamMessage

从 SSE 接收到的 EventStream 消息。

属性名	类型	说明
event	string	EventStream 的事件类型。
data	string	EventStream 的事件数据。

## 4.12 Json2plotBlock

json2plot 类型消息块

属性名	类型	说明
type	BlockType.Json2plot	消息块的类型。
content	ChartDataSchema	消息块的内容。

## 4.13 ChartDataSchema

## 从json2plot 拿到的图详情

属性名	类型	说明
dimensions	Array<Dimension>	维度列表
measures	Array<Measure>	度量列表
rows	Array<Record<string, any>>	数据行列表
chartType	EChatType	图表类型
title	string	标题

## 4.14 Dimension

charts维度数据定义

属性名	类型	说明
name	string	字段名称
displayName	string	显示名称
dataType	string	数据类型

## 4.15 Measure

属性名	类型	说明
name	string	字段名称
displayName	string	显示名称
dataType	string	数据类型
aggregation	string	聚合方式

## 4.16 ExecuteCodeBlock

执行代码进行数据分析

属性名	类型	说明

type	BlockType.ExecuteCode	消息块的类型。
content	ExecuteCodeResult	消息块的内容。

## 4.17 ExecuteCodeResult

代码执行结果的输入和输出

属性名	类型	说明
input	string	执行的代码，以 Markdown 的代码格式显示。
output	string	代码执行后转为自然语言的输出结果，以 Markdown 格式显示。

## 五、枚举

### 5.1 BlockType

消息块的类型。

值	说明
Text	文本类型
Markdown	Markdown 类型
WebSearch	联网搜索类型
Json2plot	Echarts图呈现类型
ExecuteCode	编写代码执行数据分析

### 5.2 RoleType

发送该消息的角色。

值	说明
User	用户
Assistant	AI 助手

### 5.3 EChatType

发送该消息的角色。

值	说明
Line	折线图
Column	柱状图
Pie	饼图
Circle	环形图