

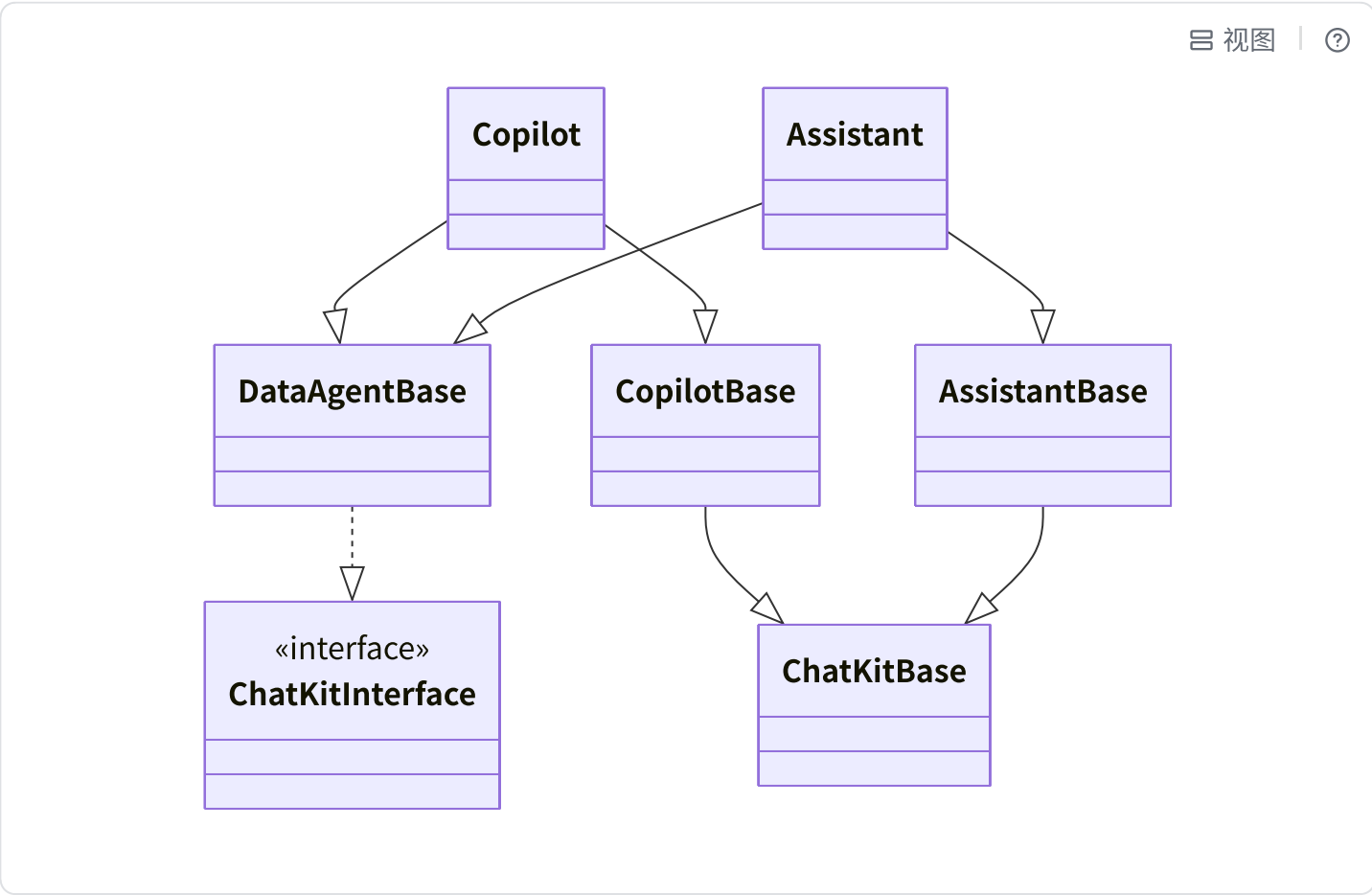
# ChatKit for DIP

## 一、基本概念

ChatKit for DIP 是专门适配 AISHU Data Agent 智能体 API 的 AI 助手。ChatKit for DIP 导出两个组件：

- Copilot：右侧跟随的 AI 助手
- Assistant：主 AI 对话入口

## 二、组件实现



**DataAgentBase** 根据 AISHU Data Agent 的 Open API 定义实现 **ChatKitInterface** 中的抽象接口。

**Copilot** 是右侧跟随的 AI 助手，继承 **DataAgentBase** 的 API 实现以及 **CopilotBase** 的交互逻辑和界面。

**Assistant** 是主 AI 对话入口，继承 **DataAgentBase** 的 API 实现以及 **AssistantBase** 的交互逻辑和界面。

## 2.1 class ChatKitDataAgent

ChatKitDataAgent 组件实现 ChatKitInterface 以下方法：

- `generateConversation()`：创建新的会话。
- `getOnboardingInfo()`：获取会话开场白信息。
- `sendMessage()`：发送消息给 AI 助手。
- `reduceAssistantMessage()`：从 EventStream 中提取出对 `action` 和 `content`，并根据 `action` 将 `content` 增量更新到 AssistantMessage。
- `shouldRefreshToken()`：判断 API 响应的状态码是否是 401，如果是，则表示需要刷新 Token。
- `terminateConversation()`：终止会话
- `getConversations()`：获取历史会话列表
- `getConversationMessages()`：加载指定历史会话消息。注意，获取会话消息后，需要对 `Messages[i].content` 进行反序列化获取消息结构。
- `deleteConversation()`：删除指定会话。

## 三、处理 EventStream

### 1、EventStream 的数据结构

EventStream 由多条 Event Message 组成，每条 Event Message 包含一个 `seq_id` 属性用于标记 Event Message 的顺序。

每一条 Event Message 都是一个 JSON 对象，表示一次对 AssistantMessage 对象的更新操作。一条 Event Message 包含 `seq_id`、`key`、`action`、`content` 四个属性：

- `seq_id`：Event Message 序号。
- `key`：要操作的 AssistantMessage 属性的路径的数组表示，需要转换为 JSONPath 后对 AssistantMessage 进行操作。例如：`["message", "content", "middle_answer", "progress", 0]` 转换为 JSONPath 后是 `"message.content.middle_answer.progress[0]"`。
- `action`：表示对 AssistantMessage 执行的操作动作：
  - `upsert` 表示在 JSONPath 路径插入数据
  - `append` 表示在 JSONPath 路径原有数据后追加内容，有两种情况会 append：
    - 如果 JSONPath 路径是一个数组下标，则在数组下标位置插入新的对象
    - 否则 JSONPath 路径表示 AssistantMessage 的某个文本类型的属性，在文本后追加内容

- `end` 表示 EventStream 结束
- `content`：表示要 `upsert` 或 `append` 的内容

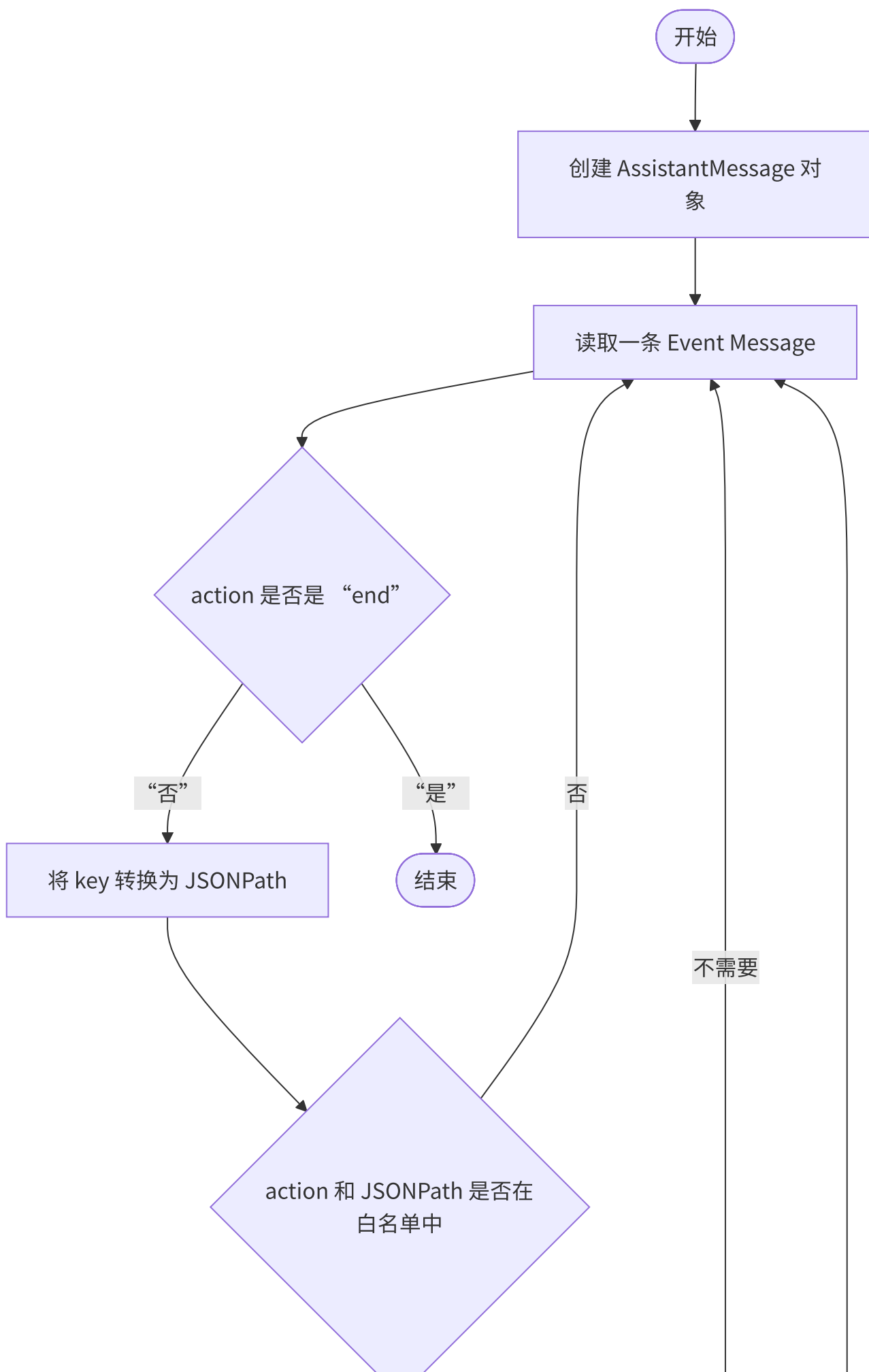
## 2、AssistantMessage 对象的数据结构

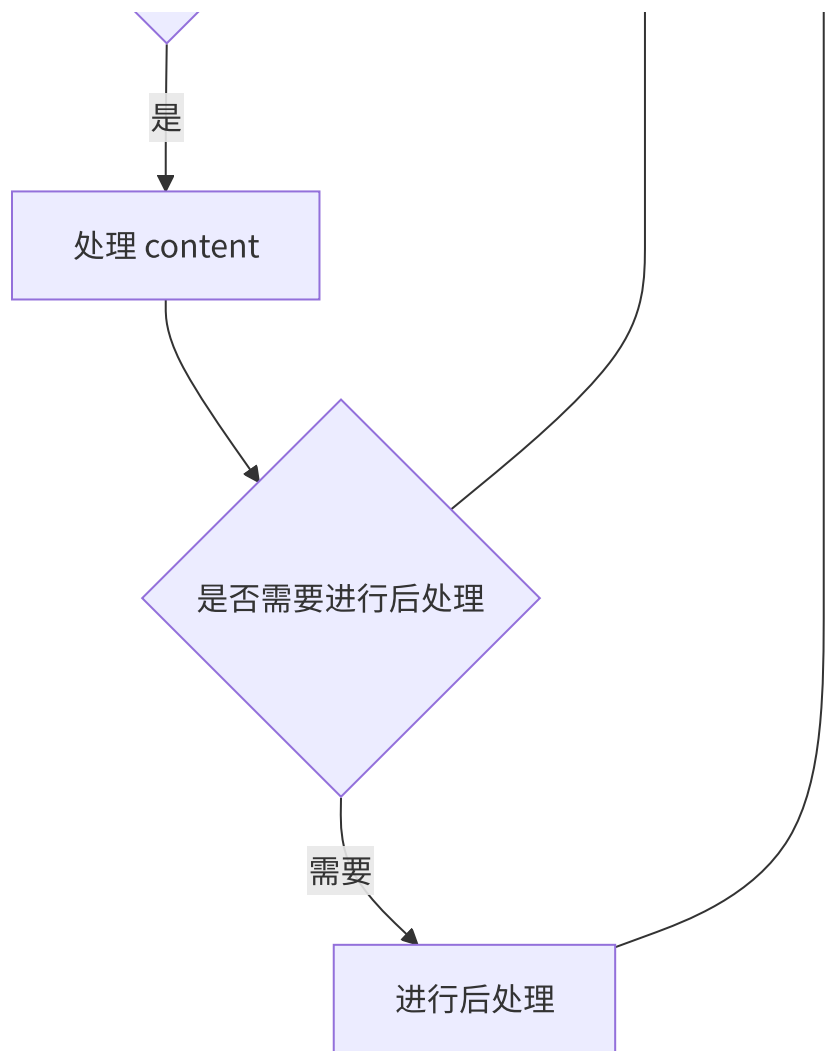
AssistantMessage 对象的数据结构与 `agent-app.schemas.yaml#/components/schemas/Message` 的定义保持一致。

## 3、处理流程

### 3.1 流程图

调用 Data Agent 的 `chat/completion` 对话接口时，接口以 EventStream 流的形式输出数据。前端需要不断接收 Event Message 并根据 `action` 来将 `content` 增量更新到 AssistantMessage 中，具体处理流程如下：





### 3.2 Event Message 白名单

判断 Event Message 包含的 action 和 JSONPath 组合是否在以下名单中：

- 如果在，则根据 action 来更新 AssistantMessage 对象，并在需要时执行后处理流程。
- 如果不在，则跳过该条 Event Message。

action	JSONPath	如何处理 content	后处理
insert	error	将 content 赋值到 JSONPath	不需要
insert	message	将 content 赋值到 JSONPath	不需要
append	message.content.middle_answer.progress[i]	将 content 赋值到 JSONPath 表示的数组下标位置	<ul style="list-style-type: none"><li>• 如果 content.stage 是 “skill”：</li></ul>

- 如果  
`content.skill_info.name`  
是 `zhipu_search_tool` , 从  
`content.answer.choices` 解  
析出 `WebSearchQuery` 结构并调  
用 `appendWebSearchBlock()`  
将 Web 搜索结果输出到界面
- 如果  
`content.skill_info.name`  
是 `json2plot` , 将  
`content.skill_info.args` 解  
析出 `ChartDataSchema` 结构并调  
用 `appendJson2plotBlock()` 将  
结果输出到界面
- 如果  
`content.skill_info.name`  
是 `execute_code` , 从  
`content.skill_info.args` 解  
析出  
`ExecuteCodeResult.input` ,  
从  
`content.answer.result.resu  
lt.stdout` 解析出  
`ExecuteCodeResult.output` ,  
然后调用  
`appendExecuteCode()` 将代码  
工具执行结果输出到界面
- 如果  
`content.skill_info.name`  
是 `text2sql` , 从  
`content.skill_info.args` 解  
析出 `Text2sqlResult` , 然后调  
用 `appendText2sql()` 结果输出  
到界面
- 如果  
`content.skill_info.name`  
是 `text2metric` , 从  
`content.skill_info.args` 解  
析出 `text2metricResult` , 然  
后调用 `appendText2metric()`  
结果输出到界面

			<ul style="list-style-type: none"><li>如果 <code>content.skill_info.name.toLowerCase()</code> 是 “search_memory”，<code>“_date”</code>，<code>“build_memory”</code> 直接忽略，不输出到界面</li><li>否则将 <code>content.skill_info.name</code> 输出到界面</li><li>如果 <code>content.stage</code> 是 “llm”，调用 <code>appendMarkdownBlock(message.content.middle_answer.progress[i].answer)</code> 将内容输出到界面</li></ul>
<code>append</code>	<code>message.content.middle_answer.progress[i].answer</code>	将 content 追加到 JSONPath 现有内容后	<ul style="list-style-type: none"><li>调用 <code>appendMarkdownBlock(message.content.middle_answer.progress[i].answer)</code> 将内容输出到界面</li></ul>

## 4、示例

### 4.1 插入对象

操作前：

代码块

```
1  {}
```

Event Message：

代码块

```
1  {
2    "seq": 0,
3    "key": ["message"],
4    "action": "upsert",
5    "content": {
6      "content": {
7        "middle_answer": {
8          "progress": []
9        }
10   }
```

```
11     }
12 }
```

操作后：

代码块

```
1  {
2    "message": {
3      "content": {
4        "middle_answer": {
5          "progress": []
6        }
7      }
8    }
9  }
```

## 4.2 追加到数组

操作前：

代码块

```
1  {
2    "message": {
3      "content": {
4        "middle_answer": {
5          "progress": []
6        }
7      }
8    }
9  }
```

Event Message：

代码块

```
1  {
2    "seq": 2,
3    "key": ["message", "content", "middle_answer", "progress", 0],
4    "action": "append",
5    "content": {
6      "stage": "llm",
7      "answer": "我来帮您"
8    }
9  }
```



```
9 }
```

操作后：

代码块

```
1  {
2    "message": {
3      "content": {
4        "middle_answer": {
5          "progress": [
6            {
7              "stage": "llm",
8              "answer": "我来帮您"
9            }
10         ]
11       }
12     }
13   }
14 }
```

## 四、解析历史对话消息

读取指定会话 ID 的消息列表后，如果匹配到 `origin` 为 `assitant` 的消息，按以下流程解析处理内容：

1. 首先需要对 `content` 进行JSON 反序列化，解析出 `middle_answer`。
2. 遍历 `middle_answer.progress` 数组，对其中每个元素按照 Event Message 白名单表格中的“后处理”流程渲染到界面。