ChatKitDataAgent

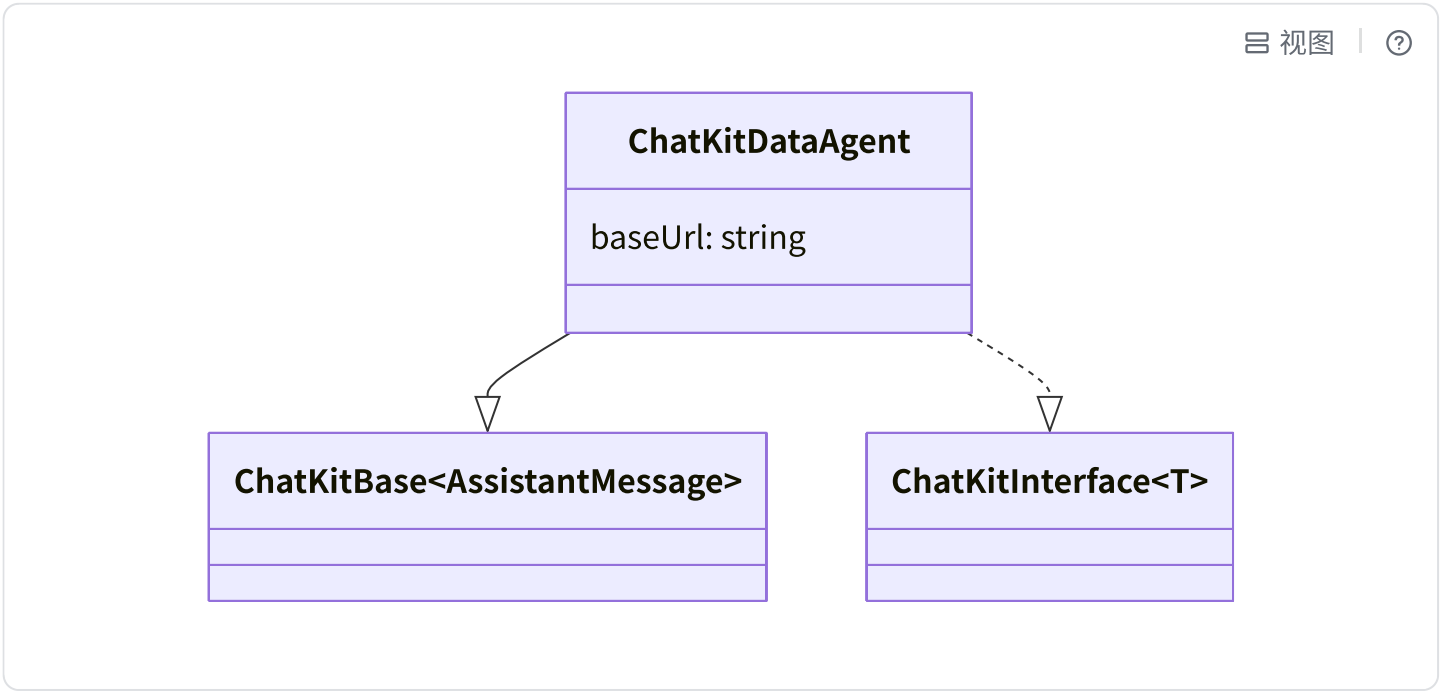
一、基本概念

ChatKitDataAgent 组件是专门适配 AISHU ADP 平台 Data Agent 智能体 API 的对话组件。

二、组件实现

2.1 class ChatKitDataAgent

ChatKitDataAgent 实现 ChatKitInterface 中的接口定义，并继承 ChatKitBase 类。



属性

属性名	类型	说明
`baseUrl`	`string`	AISHU Data Agent 平台的访问域名，默认为：/

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。

- `reduceResponseFromEventStream()`：从 EventStream 中提取出对 AssistantMessage 的操作并增量更新到 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。

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 Mesage 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，例如：`["message", "content", "middle_answer", "progress", 0]` 表示结果对象的 `"message.content.middle_answer.progress[0]"`，可以操作以下 AssistantMessage 对象的“大模型很大”。

代码块

```
{
  "message": {
    "content": {
      "middle_answer": {
        "progress": ["大模型很大"]
      }
    }
  }
}
```

- `action`：表示对 AssistantMessage 执行的操作动作：
  - `action: "upsert"` 表示在 key 路径插入数据
  - `action: "append"` 表示在 key 路径原有数据后追加内容，有两种情况会 append：
    - 如果 key 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 key 路径表示 AssistantMessage的某个文本类型的属性，在文本后追加内容
  - `action: "end"` 表示 EventStream 结束
- `content`：表示要 `upsert` 或 `append` 的内容

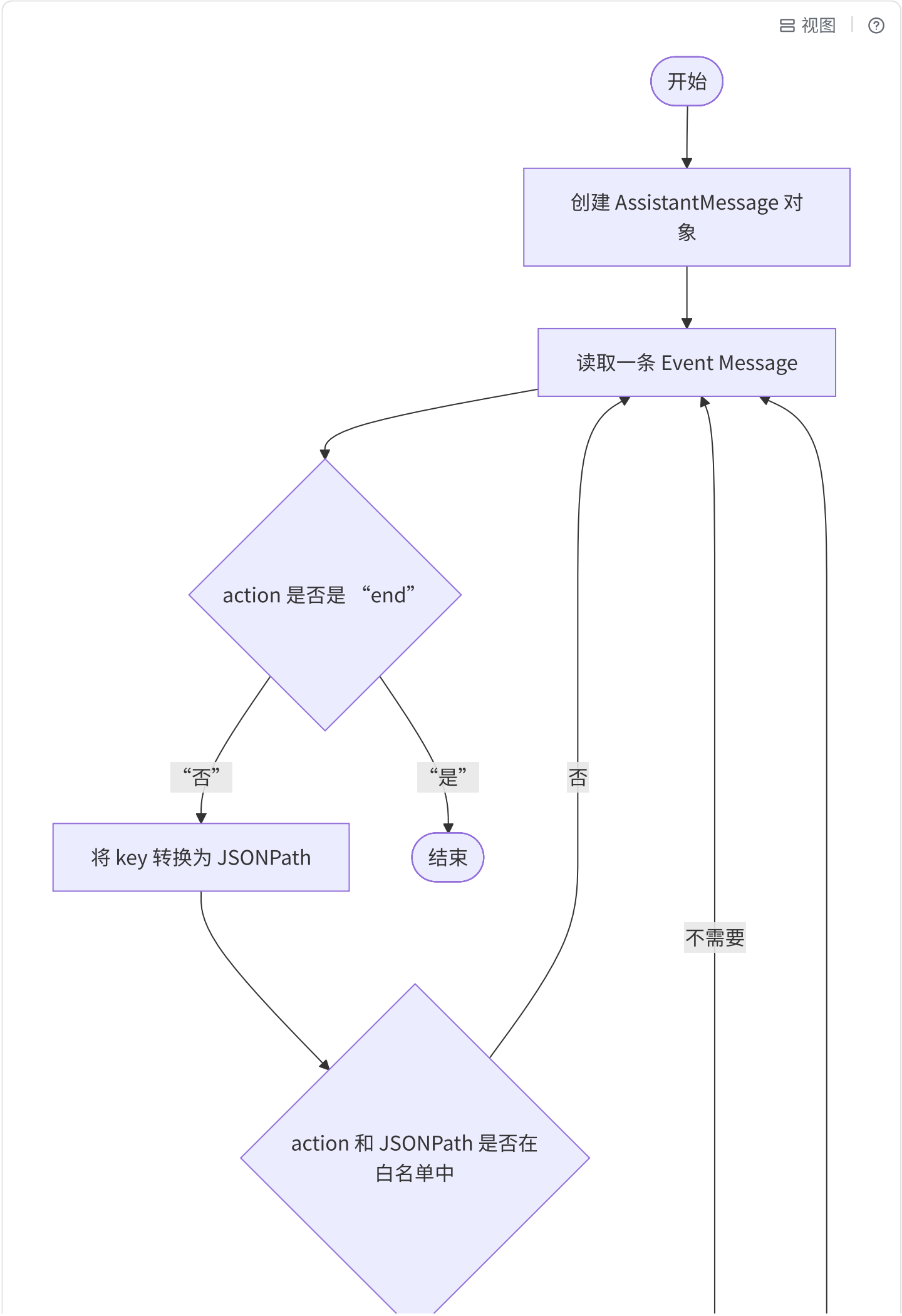
3、AssistantMessage 对象的数据结构

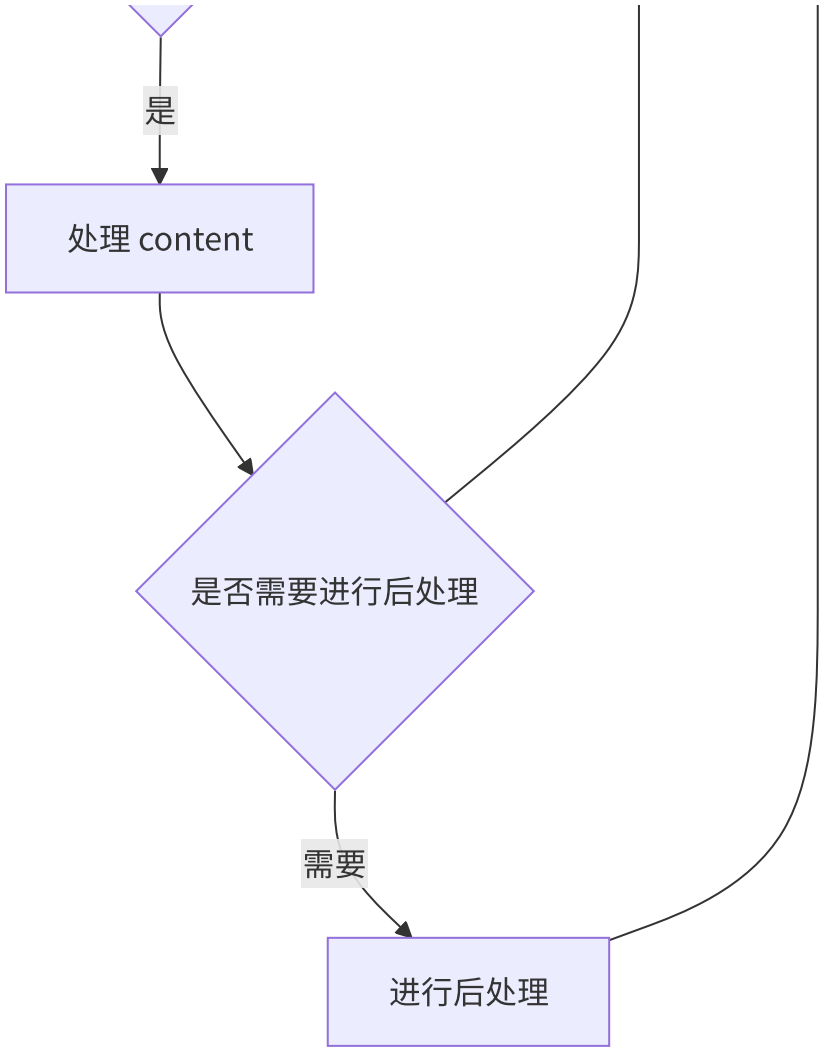
AssistantMessage 对象的数据结构与 `agent-app.schemas.yaml#/components/schemas/Message` 的定义保持一致。

4、处理流程

1.1 流程图

调用 Data Agent 的对话接口时（例如：`chat/completion`），接口以 EventStream 流的形式输出数据。前端需要不断接收 Event Message 并拼装到 AssistantMessage 中：


视图
开始
创建 AssistantMessage 对象
读取一条 Event Message
action 是否是 “end”
“否”
“是”
否
将 key 转换为 JSONPath
结束
不需要
action 和 JSONPath 是否在白名单中



1.2 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，并在需要时进行后处理流程。
- 如果不在，则跳过该条 Event Message。

action	JSONPath	如何处理 content	后处理
`upsert`	`error`	赋值到 JSONPath	不需要
`upsert`	`message`	赋值到 JSONPath	不需要
`append`	`message.content.final_answer.answer.text`	追加到 JSONPath 现有内容后	调用 `appendTextBlock()` 将内容输出到界面

`append`	`message.content.middle_answer.progress[i]`	赋值到 JSONPath 表示的数组下标	• 如果 `content.stage` 是“`skill`”： ◦ 如果 `content.skill_info.name` 是 `zhipu_search_tool`，调用 `appendWebSearchBlock()` 将 Web 搜索结果输出到界面 ◦ 否则将 `content.skill_info.name` 输出到界面 • 如果 `content.stage` 是“`llm`”，调用 `appendTextBlock()` 将 `message.content.middle_answer.progress[i].answer` 内容输出到界面
`append`	`message.content.middle_answer.progress[i].answer`	追加到 JSONPath 现有内容后	• 调用 `appendTextBlock()` 将 `message.content.middle_answer.progress[i].answer` 内容输出到界面

4、示例

4.1 插入对象

操作前：

代码块

```
{}
```

Event Message：

代码块

```
{
  "seq": 0,
  "key": ["message", "answer"],
  "action": "upsert",
  "content": "大模型"
}
```

操作后：

```
代码块
1  {
2    "message": {
3      "answer": "大模型"
4    }
5  }
```

4.2 追加对象

操作前：

```
代码块
1  {
2    "message": {
3      "answer": "大模型"
4    }
5  }
```

Event Message：

```
代码块
1  {
2    "seq": 1,
3    "key": ["message", "answer"],
4    "action": "append",
5    "content": "很大"
6  }
```

操作后：

```
代码块
1  {
2    "message": {
3      "answer": "大模型很大"
4    }
5  }
```