DSL 语言规范文档

1. 引言

本DSL（领域特定语言）设计用于描述交互式客服机器人的对话流程。通过使用简单易懂的语法，用户可以编写基于状态机的自动应答脚本，支持根据用户输入在不同模式下进行条件判断、响应输出、模式切换及变量赋值等操作。

2. 语言概述

DSL 脚本通过状态模式（MODE）来管理对话流程。每个模式下可以定义多条 `if`、`elif` 和 `else` 语句，用于处理用户输入，并根据条件选择不同的响应或行为。

语言特点：

- 基于状态的模式切换。
- 简单的条件判断（`if` / `elif` / `else`）。
- 支持输出响应、变量赋值和模式跳转。

3. 关键字与运算符

DSL 语言的基本元素包括关键字、运算符、标识符和常量。以下是语言中常用的元素：

3.1 关键字

- **start**: 定义对话开始的初始状态。
- **end**: 定义对话结束。
- **if**: 用于条件判断，后接条件和操作。
- **elif**: 在 `if` 之后用于进一步的条件判断。
- **else**: 在没有其他条件匹配时执行的操作。
- **response**: 用于输出响应。
- **go**: 用于模式跳转，指定下一个模式。
- **set**: 用于变量赋值操作。
- **in**: 检查某个值是否包含在列表或字符串中。
- **user_input**: 引用用户输入的内容。

3.2 运算符

- **=**: 赋值运算符，用于变量赋值。

- **+**: 加法运算符，用于数字的加法操作。

3.3 标识符

- **MODE**: 以大写字母组成的模式名称（如： `INIT`，`ACCOUNT` 等）。
- **ID**: 变量名或其他标识符，通常由字母和下划线组成，支持字母数字组合。

4. 语法规则（BNF）

以下是DSL语言的文法定义，采用BNF范式描述：

```
<program> ::= <start_mode> <mode_section>* <end>

<start_mode> ::= "start" <mode_name> <statement>*

<end> ::= "end"

<mode_section> ::= <mode_name> <statement>*

<mode_name> ::= [A-Z][A-Z]*  // 模式名称是大写字母

<statement> ::= <if_statement>
              | <elif_statement>
              | <else_statement>
              | <response_statement>
              | <go_statement>
              | <set_statement>

<if_statement> ::= "if" <condition> "then" <response>

<elif_statement> ::= "elif" <condition> "then" <response>

<else_statement> ::= "else" <response>

<response_statement> ::= "response" <string>

<go_statement> ::= "go" <mode_name>

<set_statement> ::= "set" <variable> "=" <expression>

<condition> ::= <expression> "in" <user_input> | <expression>
<expression> ::= <ID> | <number> | <user_input> | <string>
<variable> ::= <ID>

<user_input> ::= "user_input"
<number> ::= [0-9]+

<string> ::= '"' [^"]* '"'
```

4.1 语法解释

- : 整个脚本文件，由一个 `start` 模式开始，后面跟着多个模式部分，最后以 `end` 结束。
- **<mode_name>**: 定义每个模式的名称，如 `INIT` 、 `ACCOUNT` 等。
- : 每个模式中的基本语句，可以是条件判断、响应输出、模式跳转或变量赋值等。
- : 判断条件，可以是 `in` 操作符检查 `user_input` 中是否包含某个值，或其他表达式。
- **<response_statement>**: 输出响应文本，包含对用户输入的回复。

- **<go_statement>**: 在模式间切换。
- **<set_statement>**: 变量赋值操作，允许根据用户输入进行动态变化。

为了清晰地描述您的AST结构，并为其提供一个范式，我们将基于您给出的AST示例定义一种通用的结构。这种结构将涵盖各类节点（如 `mode`、`if`、`elif`、`else`、`response`、`go` 和 `set`）的通用格式，并将每个节点的内容具体化为规则。以下是对您所提供的AST结构的详细范式描述。

5. AST 范式定义

1.1 根节点

根节点为 `program` 类型，它包含一个 `statements` 数组，表示整个对话脚本的语句。

```
{
  "type": "program",
  "statements": [
    // 具体的语句列表，包含多种类型的节点
  ]
}
```

1.2 `mode` 节点

每个 `mode` 节点表示一个对话模式。模式名称是一个字符串（例如 "INIT"、"ACCOUNT" 等），并且每个模式下可以包含多个语句。每个 `mode` 节点都包含一个 `statements` 字段，它是该模式下所有语句的集合。

```
{
  "type": "mode",
  "mode": "<mode_name>",  // 模式名称，如 "INIT"、"ACCOUNT" 等
  "statements": [         // 当前模式下的语句列表
    // 语句内容
  ]
}
```

1.3 `if` / `elif` / `else` 节点

这些节点用于条件判断，根据用户输入执行不同的逻辑。每个 `if`、`elif` 或 `else` 节点包含条件和响应，并可能包括后续语句。节点的主要字段包括：

- `condition`： 一个数组，表示触发该条件的输入（例如 "你好"、"账户"）。
- `response`： 机器人的回应文本。

- `next_statements`：一个数组，表示当前条件满足时需要执行的后续语句（例如 `go` 跳转或变量赋值等）。

```
{
  "type": "<statement_type>",  // 'if' / 'elif' / 'else'
  "condition": ["<condition>"], // 条件，可能是一个字符串或多个字符串
  "response": "<response_text>", // 响应内容
  "next_statements": [          // 后续语句
    // 后续的语句内容，如 'go', 'set' 等
  ]
}
```

1.4 `go` 节点

`go` 节点表示模式跳转。每个 `go` 节点包含一个 `mode` 字段，表示要跳转到的目标模式。

```
{
  "type": "go",
  "mode": "<target_mode>"  // 目标模式名称，如 "ACCOUNT", "GOODS", 等
}
```

1.5 `set` 节点

`set` 节点用于设置变量的值。它包含 `variable` 字段，表示要设置的变量名，以及 `expression` 字段，表示为变量赋值的表达式。

```
{
  "type": "set",
  "variable": "<variable_name>",  // 变量名称
  "expression": {                  // 表达式，可能是简单值或运算
    "type": "<expression_type>",   // 表达式类型，如 'addition'（加法）等
    "left": "<left_operand>",      // 左操作数
    "right": "<right_operand>"     // 右操作数
  }
}
```

1.6 `response` 节点

`response` 节点用于输出机器人的回答。它仅包含 `response` 字段，表示要输出的文本。

```
{
  "type": "response",
  "response": "<response_text>"  // 响应内容
}
```

1.7 `condition` 和 `expression`

`condition` 和 `expression` 是两个重要的部分，分别表示判断条件和计算表达式。表达式可以是数值、变量、用户输入、或运算（如加法等）。条件通常是对用户输入的判断（如是否包含某个关键词）。

- `condition`：检查某个值是否在用户输入中（使用 `in`）。
- `expression`：运算表达式，用于进行简单的数学计算或变量操作。

```
{
  "type": "condition",  // "in" 或直接的字符串条件
  "left": "<expression>", // 左侧表达式（例如字符串或变量）
  "right": "<user_input>" // 用户输入
}
```

2. 总结

- **program**：表示整个对话脚本的根节点，包含多个模式。
- **mode**：

表示每个对话模式，包含多个条件判断和响应。

- **if/elif/else**：条件判断节点，根据用户输入的不同，做出不同的反应。
- **response**：机器人回应用户的文本。
- **go**：模式跳转，表示从一个模式跳转到另一个模式。
- **set**：设置变量的值，通常用在条件判断后需要更改状态时。

5. 示例脚本

以下是一个示例脚本，展示了如何使用DSL语言定义客服机器人的对话逻辑。

```
start INIT
    if "你好" in user_input then
        response "您好，很高兴为您服务，请问您的需要是"
    elif "账户" in user_input then
        response "已转移至账户模式"
        go ACCOUNT
    elif "商品" in user_input then
        response "已转移至商品模式"
        go GOODS
    else
        response "抱歉，我没有理解您的问题"

ACCOUNT
    if "余额" in user_input then
        response "您的余额为 "
    elif "充值" in user_input then
        response "请输入您所充值的金额"
        set balance = balance + user_input
    elif "退出" in user_input then
        response "您已退出账户模式"
        go INIT
    else
        response "抱歉，我没有理解您的问题"

GOODS
    if "名称" in user_input then
        response "在售商品的名称为：商品A，商品B，商品C"
    elif "查询" in user_input then
        response "已转移至查询模式，输入对应商品名称查询信息"
        go QUERY
    elif "退出" in user_input then
        response "您已退出商品模式"
        go INIT
    else
        response "抱歉，我没有理解您的问题"

QUERY
    if "商品A" in user_input then
        response "商品A：价格：100元，库存：50件"
    elif "商品B" in user_input then
        response "商品B：价格：200元，库存：30件"
    elif "商品C" in user_input then
        response "商品C：价格：300元，库存：20件"
    elif "退出" in user_input then
        response "您已退出商品查询模式"
    else
```

```
        response "抱歉，我没有理解您的问题"
end
```

对应的ast结构：

```
{
  "type": "program",
  "statements": [
    {
      "type": "mode",
      "mode": "INIT",
      "statements": [
        {
          "type": "if",
          "condition": ["你好"],
          "response": "您好，很高兴为您服务，请问您的需要是",
          "next_statements": []
        },
        {
          "type": "elif",
          "condition": ["账户"],
          "response": "已转移至账户模式",
          "next_statements": [
            {
              "type": "go",
              "mode": "ACCOUNT"
            }
          ]
        },
        {
          "type": "elif",
          "condition": ["商品"],
          "response": "已转移至商品模式",
          "next_statements": [
            {
              "type": "go",
              "mode": "GOODS"
            }
          ]
        },
        {
          "type": "else",
          "response": "抱歉，我没有理解您的问题",
          "next_statements": []
        }
      ]
    },
    {
      "type": "mode",
      "mode": "ACCOUNT",
      "statements": [
```

```
    {
      "type": "if",
      "condition": ["余额"],
      "response": "您的余额为 ",
      "next_statements": []
    },
    {
      "type": "elif",
      "condition": ["充值"],
      "response": "请输入您所充值的金额",
      "next_statements": [
        {
          "type": "set",
          "variable": "balance",
          "expression": {
            "type": "addition",
            "left": "balance",
            "right": "user_input"
          }
        }
      ]
    },
    {
      "type": "elif",
      "condition": ["退出"],
      "response": "您已退出账户模式",
      "next_statements": [
        {
          "type": "go",
          "mode": "INIT"
        }
      ]
    },
    {
      "type": "else",
      "response": "抱歉，我没有理解您的问题",
      "next_statements": []
    }
  ]
},
{
  "type": "mode",
  "mode": "GOODS",
  "statements": [
    {
      "type": "if",
      "condition": ["名称"],
```

```
      "response": "在售商品的名称为：商品A，商品B，商品C",
      "next_statements": []
    },
    {
      "type": "elif",
      "condition": ["查询"],
      "response": "已转移至查询模式，输入对应商品名称查询信息",
      "next_statements": [
        {
          "type": "go",
          "mode": "QUERY"
        }
      ]
    },
    {
      "type": "elif",
      "condition": ["退出"],
      "response": "您已退出商品模式",
      "next_statements": [
        {
          "type": "go",
          "mode": "INIT"
        }
      ]
    },
    {
      "type": "else",
      "response": "抱歉，我没有理解您的问题",
      "next_statements": []
    }
  ]
},
{
  "type": "mode",
  "mode": "QUERY",
  "statements": [
    {
      "type": "if",
      "condition": ["商品A"],
      "response": "商品A：价格：100元，库存：50件",
      "next_statements": []
    },
    {
      "type": "elif",
      "condition": ["商品B"],
      "response": "商品B：价格：200元，库存：30件",
      "next_statements": []
```

```
        },
        {
          "type": "elif",
          "condition": ["商品C"],
          "response": "商品C：价格：300元，库存：20件",
          "next_statements": []
        },
        {
          "type": "elif",
          "condition": ["退出"],
          "response": "您已退出商品查询模式",
          "next_statements": []
        },
        {
          "type": "else",
          "response": "抱歉，我没有理解您的问题",
          "next_statements": []
        }
      ]
    }
  ]
}
```

6. 扩展性

该DSL语言可以通过以下方式进行扩展：

- 添加更多的内建函数，如 `cin()` 、 `max()` ，等用于处理更多不同领域的问题。
- 支持更多条件运算符，如 `<` 、 `>` 、 `==` ，增强条件表达能力。
- 扩展响应类型，支持动态生成文本、与外部API交互等。