开发者指南

项目简介

本项目涉及一个用于描述和执行交互式客服对话流程的领域特定语言（DSL）。该DSL通过状态模式（`MODE`）来管理对话流程，支持条件判断、响应输出、模式切换及变量赋值等操作。项目包含三个主要模块：

1. **Lexer**：将输入的DSL脚本分解为词法单元（tokens）。
2. **Parser**：解析词法单元，生成抽象语法树（AST）。
3. **Interpreter**：解释并执行AST，进行对话流程的实际操作。

此外，系统还包括一些测试用例和示例，帮助开发者理解如何使用该DSL进行编程。

项目结构

```
dsl/
├── lexer.py       # 词法分析器，负责将DSL脚本转化为tokens
├── parser.py      # 语法解析器，负责将tokens解析成AST
├── interpreter.py # 解释器，负责执行AST并进行交互
scripts/
├── example1.dsl  # 示例DSL脚本
├── ...
tests/
├── test_lexer.py  # 词法分析器测试
├── test_parser.py  # 解析器测试
├── test_interpreter.py # 解释器测试
├── performance_test.py # 性能测试自动测试脚本
├── stress_test.py # 压力测试自动测试脚本
docs/
├── developer_guide.md # 开发者指南
├── user_manual.md # 用户手册
├── dsl_specification.md # DSL语言规范文档
photos/
├── bot.png  # 机器人头像
├── user.png  # 用户头像
ui/
├── gui.py # ui界面设计
main.py # 主程序
```

各模块说明

1. Lexer (`lexer.py`)

功能：将DSL脚本转化为词法单元（tokens）。

- **输入**：字符串形式的DSL脚本。
- **输出**：词法单元的列表，每个词法单元由元组表示，包含标记类型（type）和标记值（value）。
- **示例**：对于输入 `if "hello" in user_input then response "Hi there!"` , 输出可能为：

```
[('IF', 'if'), ('STRING', 'hello'), ('IN', 'in'), ('USER_INPUT', 'user_input'), ('THEN', '
```

2. Parser (`parser.py`)

功能：将词法单元（tokens）解析为抽象语法树（AST）。

- **输入**：来自Lexer的词法单元列表。
- **输出**：AST结构，代表DSL脚本的语法结构。AST为一个字典，包含模式、条件判断和响应等信息。
- **示例**：对于dsl语言 `if "hello" in user_input then response "Hi there!"` , 输出可能为：

```
{
    'type': 'program',
    'statements': [
        {'type': 'if', 'condition': ['hello'], 'response': 'Hi there!', 'next_statements':
    ]
}
```

3. Interpreter (`interpreter.py`)

功能：执行AST并与用户进行交互。解释器根据当前模式、用户输入和AST中的条件做出响应。

- **输入**：解析后的AST和初始状态（例如余额、用户输入等）。
- **输出**：执行结果，响应用户输入并更新状态（如余额、模式切换等）。
- **示例**：输入 `if "hello" in user_input then response "Hi there!"` ， 用户输入 `hello` ， 解释器将输出：

```
Hi there!
```

主要功能

1. DSL语法

以下是DSL语法的简要说明，包括核心语法元素：

- **模式声明（MODE）**：
  - 定义对话的不同模式，如 `INIT`，`GOODS` 等。
- **条件语句**：
  - 使用 `if`、`elif` 和 `else` 来根据条件判断用户输入，条件可以是某些字符串是否包含在 `user_input` 中。
- **响应语句**：
  - 使用 `response` 关键字定义机器人的回应消息。
- **控制语句**：
  - 使用 `go` 来切换模式，使用 `set` 来设置变量的值。

示例：

```
start
INIT
  if "hello" in user_input then response "Hi there!"
  elif "bye" in user_input then response "Goodbye!"
  else response "I didn't understand that."
  set balance = balance + 10
  go ACCOUNT
end
```

2. 核心功能描述

- **余额管理**：通过 `set` 语句调整余额。例如，`set balance = balance + 10` 会在当前余额的基础上加10。
- **模式切换**：通过 `go` 语句实现模式的切换。
- **用户输入处理**：根据用户的输入，系统会检查条件并执行对应的响应。如果输入包含特定关键字（如 `充值`），则会启动充值流程。

开发与调试

1. 开发流程

- **词法分析**：编写或修改DSL脚本后，首先运行 `lexer.py` 以确保脚本可以成功转化为tokens。
- **语法解析**：将词法单元传给 `parser.py`，确保脚本能够正确解析为AST。
- **执行**：将生成的AST传给 `interpreter.py`，启动解释器与用户进行交互。

2. 测试

在 `tests/` 目录下，有多个测试文件来确保每个模块的功能正确：

- **test_lexer.py**：测试词法分析器是否能正确分解DSL脚本。
- **test_parser.py**：测试解析器是否能正确生成AST。
- **test_interpreter.py**：测试解释器是否能正确执行AST并进行交互。

你可以使用Python的 `unittest` 或 `pytest` 来运行这些测试。

3. 扩展与定制

- **新语法支持**：你可以扩展DSL语法，添加新的关键字或语法规则。为此，需要在 `lexer.py` 中定义新的正则表达式，并在 `parser.py` 中添加对应的语法规则。
- **逻辑扩展**：在 `interpreter.py` 中，你可以根据项目需求扩展更多操作，例如增加新的条件判断类型或新增更多的响应语句。

如何使用

1. 编写一个DSL脚本，放在 `scripts/` 目录下。
2. 运行 `lexer.py` 将脚本转化为tokens。
3. 将生成的tokens传递给 `parser.py` ，生成AST。
4. 使用 `interpreter.py` 执行AST，并与用户交互。

示例运行

假设你有以下DSL脚本（保存在 `scripts/example1.dsl` ）：

```
start
INIT
  if "hello" in user_input then response "Hi there!"
  else response "I didn't understand that."
end
```

然后运行以下Python代码：

```
from dsl.lexer import Lexer
from dsl.parser import Parser
from dsl.interpreter import Interpreter

with open('scripts/example1.dsl', 'r', encoding='utf-8') as file:
    example_code = file.read()

lexer = Lexer(example_code)
tokens = lexer.tokenize()

parser = Parser(tokens)
ast = parser.parse()

interpreter = Interpreter(ast)
interpreter.run()  # 启动与用户的交互
```

运行时，用户输入 `hello` 会得到 `Hi there!` 的响应。

贡献指南

欢迎贡献代码！如果你发现任何bug或有新的功能建议，可以通过以下步骤参与：

1. Fork 本项目。
2. 创建一个新的分支（例如 `feature/add-new-keyword` ）。
3. 提交你的代码和测试。
4. 创建Pull Request（PR）并描述你的更改。

常见问题

- **Q**：如何调试脚本解析错误?
  - **A**：可以在 `parser.py` 中加入调试信息（如 `print` 语句）来检查语法解析过程。
- **Q**：如何增加新功能?
  - **A**：通过修改 `lexer.py` 、 `parser.py` 或 `interpreter.py` 来实现新的语法或逻辑功能。