RoDSL 语言解释器使用指南

徐奕璋 2022211373

1 概述

本项目实现了一个自定义描述在线客服机器人语言 RoDSL 的解释器。该解释器包括词法分析器、语法分析器、抽象语法树（AST）构建以及解释执行模块。通过该解释器，用户可以编写并执行符合定义语法规则的 RoDSL 代码。

2 安装与依赖

2.1 环境要求

Python 3.6+

2.2 依赖库

本项目依赖于 PLY (Python Lex-Yacc) 库，用于词法和语法分析。

2.3 安装步骤

克隆项目仓库

```
git clone https://github.com/yiz853793/robot_dsl.git
cd robot_dsl
```

安装依赖 使用 pip 安装 PLY：

```
pip install ply
```

3 词法分析器 (Lexer)

词法分析器负责将输入的 DSL 代码字符串分解成一系列的标记（tokens）。本项目使用 PLY 库实现词法分析。

该语言包含关键字 `if`，`begin`，`else`，`end`，`while` ，`function`，`return`，`itoa`，`atoi`，`and`，`or`，`not`，`true`，`false`。运算符包含 `+`，`-`，`*`，`/`，`=`，`==` ，`!=`，`<=`，`<`，`>`，`>=`。分隔符有 `(`，`)`，`[`，`]` 和`,`。

RoDSL 中数分为整数和浮点数，整数每个字符都是 `0` 至 `9`，浮点数有一个小数点`.`，其余每个字符都是 `0` 至 `9`。标识符由字母、数字和下划线组成，且以字母或下划线开头。字符串的开头和结尾相同，为`'` 或`"`，中间由以 utf-8 编码的字符和转移符号`\n`，`\t`，`\\`，`\'`，`\"` 构成。

RoDSL 允许调用外部 python 程序，使用方法为`./`后面紧跟 python 脚本的地址，可以是绝对地址和相对地址。

使用示例 以下程序运行后输出 a.dsl 内容的 tokens。

```
python lexer.py scripts\a.dsl
```

4 语法分析器 (Parser)

语法分析器负责将词法分析器生成的标记序列解析为抽象语法树（AST）。本项目同样使用 PLY 库实现语法分析。该语言的 BNF 为

$$
\begin{aligned}
\langle program \rangle &::= \langle program \rangle \langle statement \rangle \\
&\mid \epsilon \\
\\
\langle \text{statement} \rangle &::= \langle \text{assign} \rangle \\
&\mid \langle \text{ifState} \rangle \\
&\mid \langle \text{whileState} \rangle \\
&\mid \langle \text{pythonCall} \rangle \\
&\mid \langle \text{functionCall} \rangle \\
&\mid \langle \text{functionDef} \rangle \\
&\mid \langle \text{returnStatement} \rangle \\
\langle assign \rangle &::= \texttt{id=} \langle expression \rangle \\
&\mid \langle arrayitem \rangle \texttt{=} \langle expression \rangle \\
\langle ifState \rangle &::= \texttt{if} \langle condition \rangle \texttt{begin} \langle program \rangle \texttt{end} \\
&\mid \texttt{if} \langle condition \rangle \texttt{begin} \langle program \rangle \texttt{else} \langle program \rangle \texttt{end} \\
\langle functionDef \rangle &::= \texttt{function id (} \langle argumentLists \rangle \texttt{) begin} \langle program \rangle \texttt{end} \\
\langle functionCall \rangle &::= \texttt{id (} \langle argumentLists \rangle \texttt{)} \\
\langle returnStatement \rangle &::= \texttt{return} \\
&\mid \texttt{return} \langle condition \rangle \\
\langle condition \rangle &::= \langle condition \rangle \texttt{or} \langle boolexpression \rangle \\
&\mid \langle boolexpression \rangle \\
\langle boolexpression \rangle &::= \langle boolexpression \rangle \texttt{and} \langle boolterm \rangle \\
&\mid \langle boolterm \rangle \\
\langle boolterm \rangle &::= \langle boolfactor \rangle \\
&\mid \langle boolterm \rangle \texttt{==} \langle boolfactor \rangle \\
&\mid \langle boolterm \rangle \texttt{!=} \langle boolfactor \rangle \\
&\mid \langle boolterm \rangle \texttt{<=} \langle boolfactor \rangle \\
&\mid \langle boolterm \rangle \texttt{<} \langle boolfactor \rangle \\
&\mid \langle boolterm \rangle \texttt{>=} \langle boolfactor \rangle \\
&\mid \langle boolterm \rangle \texttt{>} \langle boolfactor \rangle \\
\langle boolfactor \rangle &::= \texttt{not} \langle boolfactor \rangle \\
&\mid \langle expression \rangle \\
\langle expression \rangle &::= \langle expression \rangle \texttt{+} \langle term \rangle \\
&\mid \langle expression \rangle \texttt{-} \langle term \rangle \\
&\mid \langle term \rangle
\end{aligned}
$$

$$
\begin{aligned}
\langle term \rangle ::= & \langle term \rangle * \langle factor \rangle \\
& | \langle term \rangle / \langle factor \rangle \\
& | \langle factor \rangle \\
\langle factor \rangle ::= & (\langle condition \rangle) \\
& | \texttt{-} \langle factor \rangle \\
& | (\texttt{atoi}) \langle factor \rangle \\
& | (\texttt{itoa}) \langle factor \rangle \\
& | \texttt{number} \\
& | \texttt{str} \\
& | \texttt{id} \\
& | \langle arrayItem \rangle \\
& | \langle array \rangle \\
& | \langle pythonCall \rangle \\
& | \texttt{true} \\
& | \texttt{false} \\
& | \langle functionCall \rangle \\
\langle pythonCall \rangle ::= & \texttt{callpy} (\langle argumentLists \rangle) \\
\langle array \rangle ::= & [\langle argumentLists \rangle] \\
\langle arrayItem \rangle ::= & \texttt{id} [\langle expression \rangle] \\
\langle argumentLists \rangle ::= & \langle argumentList \rangle \\
& | \epsilon \\
\langle argumentList \rangle ::= & \langle expression \rangle \\
& | \langle argumentList \rangle , \langle expression \rangle
\end{aligned}
$$

if 语句 `if condition begin program1 (else program2) end`，else 部分不是必须的。

while 语句 `while condition begin program1 end`

函数声明语句 `function id (argumentLists) begin program end`,其中 `argumentLists` 是参数列表，每个都必须是 `id`。

函数调用语句 `id (argumentLists)`

5 抽象语法树 (AST)

AST 记录了程序运行的拓扑结构，节点记录了操作的类型和需要执行的子节点，执行时递归操作。例如程序 `a = 1 + 1` 的 AST 为：

```
(program, None, None)
|-(assign, None, =)
  |-(ID, a, None)
  |-(expression, None, +)
    |-(NUMBER, 1, None)
    |-(NUMBER, 1, None)
```

6 解释器 (Interpreter)

解释器负责执行 AST，按照定义的语义运行 RoDSL 代码。它处理变量、函数调用、控制流等。

主要功能

变量管理: 维护变量的作用域和生命周期。

函数调用: 支持用户定义函数及内置函数（如 print, input, len）。

控制流: 执行条件语句和循环结构。

外部 Python 调用: 支持调用外部 Python 脚本。

错误处理: 提供详细的运行时错误信息。

7 使用指南

DSL 代码文件以.dsl 为后缀，在 Windows 环境下的路径为 `script_path`，通过以下命令运行 DSL 代码文件。

```
python main.py script_path
```

`script_path` 可以是绝对地址和相对地址。如 D:\dsl\scripts\a.dsl 和 scripts\a.dsl。

A 示例代码

示例 1：简单输入输出

```
# scripts/hello.dsl
print("Hello ","DSL Interpreter!\n")
# 输出 Hello, DSL Interpreter!
name = input('请问您的姓名: \n')
print('你好' + name + '\n')
a = len(name)
a = a[0]
print('您的名字有 ', a,' 个字\n')
```

输出

```
Hello DSL Interpreter!
请问您的姓名:
xyz
你好:xyz
您的名字有 3 个字
```

`print` 函数可以接收若干参数，将参数转为字符串合并后输出。`input` 函数可以接收若干参数，将参数的字符串输出后等待用户输入，返回值为用户输入的字符串。`len` 函数可以接收若干参数，计算每个参数的长度，如果为数或布尔值等没有长度的数据类型长度为 None，返回数组。

示例 2: 变量与表达式

```
# scripts/variables.dsl
a = 10
b = 20
c = a + b * 2
print(c, '\n')  # 输出 50
```

示例 3: 条件语句

```
# scripts/condition.dsl
a = 5
if a > 3 begin
    print("a 大于 3\n")
else
    print("a 小于等于 3\n")
end
```

输出:

```
a 大于 3
```

示例 4: 循环语句

```
# scripts/loop.dsl
i = 0
while i < 5 begin
    print("i =", i, '\n')
    i = i + 1
end
```

输出:

```
i = 0
i = 1
i = 2
i = 3
i = 4
```

示例 5: 函数声明与调用

```
# scripts/function.dsl
function add(x, y) begin
    return x + y
end

result = add(10, 15)
print("Result : ", result, '\n')
```

输出:

```
Result : 25
```

示例 6: 调用外部 Python 脚本

```
# pyscripts\check_remain.py
import sys

if __name__ == '__main__':
    remain = {'张三' : 100, '李四' : 200, '王五' : 300}
    if len(sys.argv) >= 2 :
        name = sys.argv[1]
        if name in remain :
            print( remain[name] )
```

```
# scripts\check_remain_bot.dsl
name = input('请问您的姓名，输入\'退出\'退出：')

while name != '退出' begin
    remain = ./pyscripts\check_remain.py(name)

    if remain
    begin
        print('您的余额为', remain, '\n')
        if (atoi)remain < 200 begin
            print('您的余额少于 200\n')
        end
    else
        print('对不起，没有查到您的余额。\n')
    end

    name = input('请问您的姓名，输入\'退出\'退出：')
end
```

输出：

```
请问您的姓名，输入' 退出' 退出：张三
您的余额为 100
您的余额少于 200
请问您的姓名，输入' 退出' 退出：李四
您的余额为 200
请问您的姓名，输入' 退出' 退出：王五
您的余额为 300
请问您的姓名，输入' 退出' 退出：王六
对不起，没有查到您的余额。
请问您的姓名，输入' 退出' 退出：退出
```

注意这里的 `pyscripts\check_remain.py` 是相对运行环境的地址，而不是相对.dsl 文件的地址，因此为避免出错使用，最好使用绝对地址。

B 扩展与贡献

欢迎对本项目进行扩展和贡献！以下是一些建议的改进方向：

- 增加更多内置函数: 如数学函数、字符串处理函数等。
- 增强错误处理: 提供更多上下文信息，支持错误恢复
- 优化性能: 提高解释器的执行效率，支持更大规模的代码。
- 扩展语言特性: 添加类与对象支持、模块化编程等高级特性。

如有兴趣，请提交 Pull Request 或 Issue。