



# 北京邮电大学

Beijing University of Posts and Telecommunications

## 《程序设计实践》

### 实验报告

题    目	:	基于领域特定脚本语言的客服机器人的设计与实现
姓    名	:	宋健
学    号	:	2022212702
班    级	:	2022211305
专    业	:	计算机科学与技术
指导老师	:	闫丹凤，赵耀

2024 年 12 月 25 日

---

- 作业描述
- 基本要求
- 评分标准
- 实验环境
- 提交文件
- 目录介绍及模块划分
  - 模块划分
- 如何运行
  - 后端
  - 前端DEMO
- 示例
  - fibonacci
    - 脚本文件
    - 运行示例
  - wether
    - 启动api server demo
    - 启动后端
    - 运行示例
  - simple
    - 后端启动方法
    - 运行示例
- 记法**
  - 介绍
  - 关键字和保留字
  - 注释
  - 类型
  - 变量
  - 表达式和运算符
  - 条件语句
  - 循环语句
  - 函数
  - 事件
  - 内置函数和事件
    - 内置事件
      - 示例
    - 内置函数
  - string 对象
  - JSON 对象
  - 示例脚本
    - 示例1：简单算术运算
    - 示例2：条件语句
    - 示例3：循环语句
    - 示例4：函数调用
    - 示例5：事件处理
  - 结论
- 设计与实现
  - 数据结构
  - 模块划分
  - 功能
- 接口
  - 程序间接口
  - 人机接口

测试

测试桩

自动测试脚本

示例脚本

示例1：简单算术运算

示例2：条件语句

示例3：循环语句

示例4：函数调用

示例5：事件处理

结论

附录

项目概述

语言统计

目录结构

详细信息

---

## 作业描述

领域特定语言（Domain Specific Language, DSL）可以提供一种相对简单的文法，用于特定领域的业务流程定制。本作业要求定义一个领域特定脚本语言，这个语言能够描述在线客服机器人（机器人客服是目前提升客服效率的重要技术，在银行、通信和商务等领域的复杂信息系统中有广泛的应用）的自动应答逻辑，并设计实现一个解释器解释执行这个脚本，可以根据用户的不同输入，根据脚本的逻辑设计给出相应的应答。

## 基本要求

- 脚本语言的语法可以自由定义，只要语义上满足描述客服机器人自动应答逻辑的要求。
- 程序输入输出形式不限，可以简化为纯命令行界面。
- 应该给出几种不同的脚本范例，对不同脚本范例解释器执行之后会有不同的行为表现。

## 评分标准

本作业考察学生规范编写代码、合理设计程序、解决工程问题等方面的综合能力。满分100分，具体如下：

- 风格**：满分15分，其中代码注释6分，命名6分，其它3分。（见源码）
- 设计和实现**：满分30分，其中数据结构7分，模块划分7分，功能8分，文档8分。（见文档）
- 接口**：满分15分，其中程序间接口8分，人机接口7分。（将源码及文档）
- 测试**：满分30分，测试桩15分，自动测试脚本15分。（见源码）
- 记法**：满分10分，文档中对此脚本语言的语法的准确描述。（见文档）

## 实验环境

- windows 11
- python 3.11
- node 22.2
- npm 10.9
- git version 2.45.2.windows.1

# 提交文件

- 报告
- 程序源代码
- 可执行文件

注意：抄袭或有意被抄袭均为0分。

# 目录介绍及模块划分

1	.	
2	├─ README.md	# 项目的总体介绍和使用说明
3	├─ docs	# 文档目录
4	└─ report.md	# 实验报告的Markdown版本
5	└─ report.pdf	# 实验报告的PDF版本
6	├─ examples	# 示例脚本目录
7	└─ eq.krl	# 示例脚本，展示了基本的事件处理
8	└─ fibonacci.krl	# 示例脚本，展示了斐波那契数列的计算
9	└─ frontend	# 前端代码目录，包含Vue.js项目的代码
10	└─ server	# wether脚本的api服务端代码目录，包含Quart服务器的
	代码	
11	└─ simple.krl	# 示例脚本
12	└─ test_data_type.krl	# 示例脚本
13	└─ weather.krl	# 示例脚本，展示了http请求的使用
14	├─ requirements.txt	# 项目的依赖包列表
15	├─ src	# 源代码目录，包含项目的主要实现代码
16	└─ api_type.py	# API类型定义
17	└─ arg.py	# 命令行参数解析
18	└─ config	# 配置目录
19	└─ gen_krl.py	# KRL生成器
20	└─ interpreter	# 解释器模块
21	└─ lex.py	# 词法分析器实现
22	└─ node	# 节点模块
23	└─ parser.out	# 语法分析器输出
24	└─ parser.py	# 语法分析器实现
25	└─ parsetab.py	# 语法分析器表
26	└─ server.py	# 后端入口
27	└─ symbols	# 符号表模块
28	└─ user.py	# 用户实现
29	└─ utils	# 通用工具模块
30	├─ tests	# 测试目录，包含项目的测试代码
31	└─ conftest.py	# 测试配置
32	└─ server	# 服务器测试目录
33	└─ test_node.py	# 节点测试实现
34	└─ test_symbols.py	# 符号表测试实现
35	└─ tmp	# 临时目录，包含临时文件

# 模块划分

- 词法分析器（lexer）：负责将输入字符串分割成标记（token）。
- 语法分析器（parser）：负责将标记序列解析成抽象语法树（AST）。
- 解释器（interpreter）：负责执行抽象语法树中的指令。

- 符号表 (symbols) : 负责存储和管理符号信息。
- 服务器 (server) : 提供API接口, 处理用户请求。
- 前端 (frontend) : 提供用户界面, 允许用户与服务器进行交互。
- 测试 (tests) : 包含项目的测试代码, 确保各个模块的正确性和稳定性。

## 如何运行

---

在准备号[实验环境](#)后,进入项目主目录。

### 后端

- 安装依赖 `pip install -r .\requirements.txt`
- 运行示例脚本(可换用其它脚本) `python .\src\server.py --file .\examples\fibonacci.krl`

成功启动后结果如下:

```
PS krl> python .\src\server.py --file .\examples\fibonacci.krl
* Serving Quart app 'server'
* Debug mode: True
* Please use an ASGI server (e.g. Hypercorn) directly in production
* Running on http://localhost:5001 (CTRL + C to quit)
```

### 前端DEMO

- 进入前端demo目录 `cd .\examples\frontend\`
- 安装依赖 `npm i`
- 启动项目 `npm run dev`
- 从web端进入 `http://localhost:5173/` (端口号可能会变化)

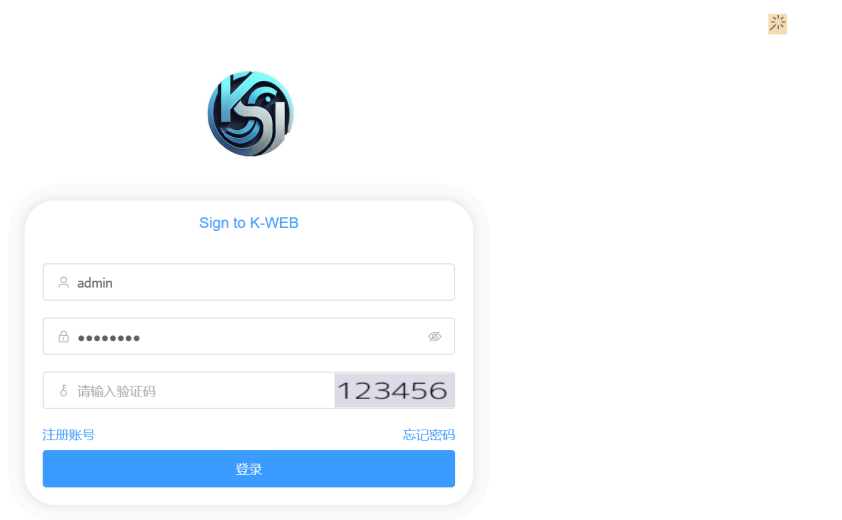
启动成功后界面如下:

```
PS frontend> npm run dev

> vue-template@0.0.0 dev
> vite

VITE v5.4.11 ready in 2493 ms

→ Local:   http://localhost:5173/
→ Network: use --host to expose
→ press h + enter to show help
```



## 示例

- 账号: admin
- 密码: 123456

## fibonacci

后端启动方式

```
1 | python .\src\server.py --file .\examples\fibonacci.kr1
```

## 脚本文件

```
1 | # 计算斐波那契数列
2 | event "start" {
3 |     print("你好! 欢迎使用我们的斐波那契数列计算机人。");
4 |     int n = get("请输入你想计算的斐波那契数列的项数: ");
5 |     int result = fbc(n);
6 |     print("第 ${n} 项的斐波那契数列结果是: ${result}");
7 |     ask_again();
```

```

8   }
9
10  event "end" {
11      print("感谢你的使用，再见！");
12  }
13  event "other" {
14      print("无其它功能，我们还是来算算斐波那契数列吧!");
15      ask_again();
16  }
17
18  fn ask_again() {
19      string choice = get("你想继续计算吗？(是/否)");
20      if (choice == "是") {
21          int n = get("请输入你想计算的斐波那契数列的项数：");
22          int result = fbc(n);
23          print("第 ${n} 项的斐波那契数列结果是：${result}");
24          ask_again();
25      } else if(choice == "否") {
26          exit();
27      } else {
28          print("小淘气！不要乱输入哟~");
29      }
30  }
31
32  fn fbc(int n) {
33      if (n == 0 || n == 1) {
34          return 1;
35      }
36      return fbc(n - 1) + fbc(n - 2);
37  }

```

## 运行示例

同时多个用户运行



# wether

本示例包含http请求示例

## 启动api server demo

```
python .\examples\server\src\main.py
```

脚本示例

```
1  from quart import Quart, request, jsonify
2
3  app = Quart(__name__)
4
5  # 存储数据的简单字典
6  data_store = {}
7
8
9  @app.route('/data', methods=['POST'])
10 async def post_data():
11     data = await request.json
12     print("data is :",data)
13     key = data.get('key')
14     value = data.get('value')
15     if key and value:
16         data_store[key] = value
17         return jsonify({"message": "Data stored successfully"}), 201
18     else:
19         return jsonify({"error": "Invalid data"}), 201
20
21 @app.route('/data/<key>', methods=['GET'])
22 async def get_data(key):
23     value = data_store.get(key)
24     if value:
25         return jsonify({key: value}), 200
26     else:
27         return jsonify({"error": "Not found"}), 404
28
29 @app.route('/weather', methods=['GET'])
30 async def weather():
31     return jsonify({"climate": "好", "temp": "适宜"}), 200
32
33
34 if __name__ == '__main__':
35     app.run(host='0.0.0.0', port=5000, debug=True)
```

## 启动后端

```
1 python .\src\server.py --file .\examples\weather.kr1
```

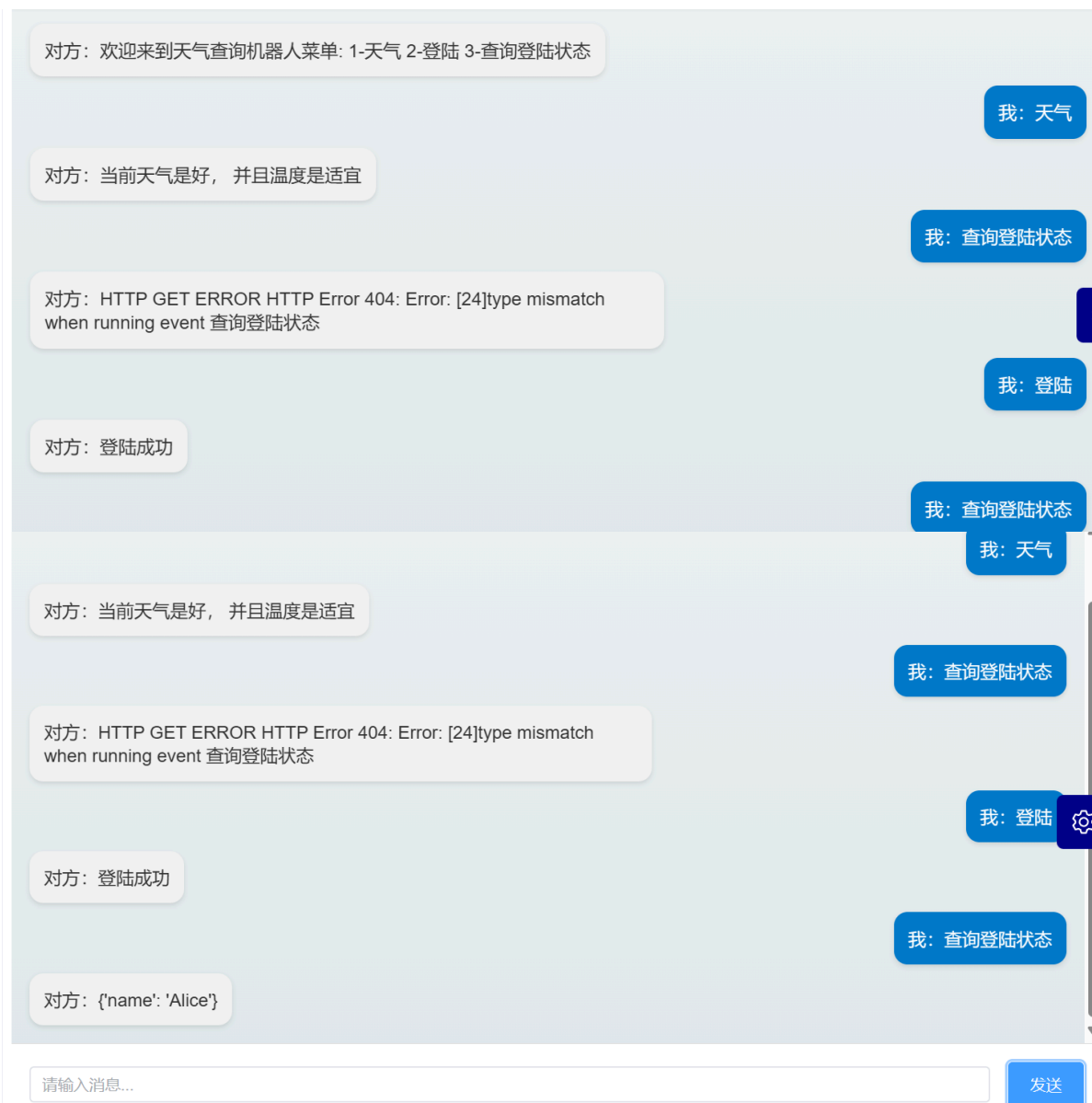
脚本

```
1 event "start" {
2     print("欢迎来到天气查询机器人");
3     menu();
```



```
4 }
5 event "天气" {
6     json res = hget("http://localhost:5000/weather");
7     string climate = res.climate;
8     string temp = res.temp;
9     print("当前天气是${climate}, 并且温度是${temp}");
10 }
11 event "登陆" {
12     string post_url = "http://127.0.0.1:5000/data";
13     string name = "Alice";
14     string key = "name";
15     json post_data = {
16         "key": key,
17         "value": name
18     };
19     hpost(post_url, post_data);
20     print("登陆成功");
21 }
22 event "查询登陆状态" {
23     string get_url = "http://127.0.0.1:5000/data/name";
24     json res = hget(get_url);
25     print("${res}");
26 }
27 event "other" {
28     print("没有这个功能(☹️)");
29     menu();
30 }
31 event "退出" {
32     exit();
33 }
34 fn menu() {
35     print("菜单:\n");
36     print(" 1-天气");
37     print(" 2-登陆");
38     print(" 3-查询登陆状态");
39 }
```

## 运行示例



## simple

### 后端启动方法

```
1 python .\src\server.py --file .\examples\simple.kr1
```

脚本:

```
1 event "start" {
2     print("欢迎来到年龄裁断机器人\n");
3     int age = get("please input your age:\n");
4     check_age(age);
5 }
6
7 fn check_age(int age) {
8     if (age < 0) {
9         print("Age cannot be negative. Please try again.");
10        int new_age = get("please input your age:\n");
11        check_age(new_age);
12    } else if (age < 18) {
```

```
13     print("You are a minor.");
14 } else if (age < 65) {
15     print("You are an adult.");
16 } else {
17     print("You are a senior.");
18 }
19 }
20
21 event "再来一次" {
22     int age = get("小馋猫, please input your age:\n");
23     check_age(age);
24 }
25
26 event "退出" {
27     exit();
28 }
29
30 event "other" {
31     print("菜单:");
32     print("1-再来一次");
33     print("2-退出");
34 }
```

运行示例

LOGO

首页

关于

客户机器人 DSL DEMO 聊天助手

对方：欢迎来到年龄判断机器人 please input your age:

我：16

对方：You are a minor.

我：14

对方：菜单:1-再来一次2-退出

我：再来一次

对方：小馋猫, please input your age:

我：ff

请输入消息...

发送

LOGO

首页

关于

客户机器人 DSL DEMO 聊天助手

对方：Error: invalid literal for int() with base 10: 'ff' when running event 再来一次

我：ff

对方：菜单:1-再来一次2-退出

我：ff

我：再来一次

对方：小馋猫, please input your age:

我：18

对方：You are an adult.

请输入消息...

发送

LOGO

首页

关于

客户机器人 DSL DEMO 聊天助手

对方：小馋猫, please input your age:

我：再来一次

对方：You are an adult.

我：18

对方：小馋猫, please input your age:

我：90

对方：You are a senior.

请输入消息...

发送

# 记法

## 介绍

本客户机器人 DSL（领域特定语言）旨在描述在线客服机器人的自动应答逻辑。该语言提供了一种简单且直观的方式来定义事件、条件、循环、函数和变量，以便根据用户输入生成相应的响应。

## 关键字和保留字

DSL中的关键字和保留字用于定义语言的基本结构和控制流。以下是主要的关键字和保留字：

- **event**：定义事件。
- **if**：条件语句。
- **else**：条件语句的分支。
- **while**：循环语句。
- **return**：返回语句。
- **fn**：定义函数。

## 注释

DSL支持单行注释，使用 `#` 开头。注释用于在代码中添加说明和解释。

```
1 | # 这是一个注释
```

## 类型

DSL支持以下基本类型：

- **int**：整数类型。
- **float**：浮点数类型。
- **string**：字符串类型。
- **json**：JSON对象类型。

## 变量

变量用于存储数据，可以在表达式中使用。变量的定义和赋值如下：

```
1 | int x = 5;  
2 | float y = 3.14;  
3 | string name = "Alice";  
4 | json data = {"key": "value"};
```

## 表达式和运算符

DSL支持以下运算符：

- **算术运算符**：`+`、`-`、`*`、`/`
- **关系运算符**：`==`、`!=`、`<=`、`>=`、`<`、`>`
- **逻辑运算符**：`&&`、`||`、`!`

- 赋值运算符: `=`

## 条件语句

条件语句用于根据条件执行不同的代码块。DSL支持 `if` 和 `else` 语句:

```
1 int x = 5;
2 if (x == 5) {
3     print("x is 5");
4 } else {
5     print("x is not 5");
6 }
```

## 循环语句

循环语句用于重复执行代码块。DSL支持 `while` 语句:

```
1 int x = 0;
2 while (x < 10) {
3     print(x);
4     x = x + 1;
5 }
```

## 函数

函数用于封装可重用的代码块。函数的定义和调用如下:

```
1 fn factorial(int n) {
2     if (n == 0) {
3         return 1;
4     } else {
5         return n * factorial(n - 1);
6     }
7 }
8
9 int result = factorial(5);
10 print("Factorial of 5 is ${result}");
```

## 事件

事件用于定义特定的触发条件和响应逻辑, 输入对应的关键字能够触发事件 (未来引入AI可以从自然语言中提取关键字)。事件的定义如下:

```
1 event "start" {
2     print("welcome to the chatbot!");
3     int choice = get("Enter 1 for weather, 2 for news:");
4     if (choice == 1) {
5         trigger("weather");
6     } else if (choice == 2) {
7         trigger("news");
8     } else {
9         print("Invalid choice");
10    }
```

```

11 }
12
13 event "weather" {
14     print("The weather is sunny.");
15 }
16
17 event "news" {
18     print("Today's news: ...");
19 }

```

## 内置函数和事件

DSL提供了一些内置函数和事件，用于处理常见的操作和事件。

### 内置事件

- **start**: 启动事件，在脚本开始执行时触发。
- **end**: 结束事件，在脚本结束执行时触发。
- **other**: 其他事件，用于处理未定义的事件。

### 示例

```

1  event "start" {
2      print("欢迎来到年龄判断机器人\n");
3      int age = get("please input your age:\n");
4      check_age(age);
5  }
6
7  fn check_age(int age) {
8      if (age < 0) {
9          print("Age cannot be negative. Please try again.");
10         int new_age = get("please input your age:\n");
11         check_age(new_age);
12     } else if (age < 18) {
13         print("You are a minor.");
14     } else if (age < 65) {
15         print("You are an adult.");
16     } else {
17         print("You are a senior.");
18     }
19 }
20
21 event "再来一次" {
22     int age = get("小馋猫，please input your age:\n");
23     check_age(age);
24 }
25
26 event "退出" {
27     exit();
28 }
29
30 event "other" {
31     print("菜单:");
32     print("1-再来一次");
33     print("2-退出");

```

## 内置函数

- **print**: 输出字符串到控制台。

```
1 | print("Hello, world!");
```

- **get**: 获取用户输入, 返回字符串。

```
1 | string name = get("Please enter your name:");
```

- **hget**: 发送HTTP GET请求, 返回响应字符串。

```
1 | string response = hget("https://api.example.com/data");
```

- **hpost**: 发送HTTP POST请求, 返回响应字符串。

```
1 | json data = {"key": "value"};
2 | string response = hpost("https://api.example.com/data", data);
```

- **exit**: 退出脚本执行。

```
1 | exit();
```

## string 对象

支持format

```
1 | int n = get("请输入你想计算的斐波那契数列的项数: ");
2 | int result = fbc(n);
3 | print("第 ${n} 项的斐波那契数列结果是: ${result}");
```

## JSON 对象

DSL支持JSON对象, 用于存储结构化数据。JSON对象的定义如下:

```
1 | json data = {
2 |     "name": "Alice",
3 |     "age": 30,
4 |     "address": {
5 |         "city": "Wonderland",
6 |         "street": "Rabbit Hole"
7 |     }
8 | };
```

注: 本脚本不支持完整的JSON 对象的所有功能, 仅支持部分功能



## 示例脚本

以下是几个示例脚本，展示了DSL的基本语法和功能：

### 示例1：简单算术运算

```
1 int x = 3 + 4 * 10;
2 print(x);
```

### 示例2：条件语句

```
1 int x = 5;
2 if (x == 5) {
3     print("x is 5");
4 } else {
5     print("x is not 5");
6 }
```

### 示例3：循环语句

```
1 int x = 0;
2 while (x < 10) {
3     print(x);
4     x = x + 1;
5 }
```

### 示例4：函数调用

```
1 fn factorial(int n) {
2     if (n == 0) {
3         return 1;
4     } else {
5         return n * factorial(n - 1);
6     }
7 }
8
9 int result = factorial(5);
10 print("Factorial of 5 is ${result}");
```

### 示例5：事件处理

```
1 event "start" {
2     print("welcome to the chatbot!");
3     int choice = get("Enter 1 for weather, 2 for news:");
4     if (choice == 1) {
5         trigger("weather");
6     } else if (choice == 2) {
7         trigger("news");
8     } else {
9         print("Invalid choice");
10    }
11 }
```

```
12
13   event "weather" {
14       print("The weather is sunny.");
15   }
16
17   event "news" {
18       print("Today's news: ...");
19   }
```

## 结论

本DSL通过提供简单且直观的语法，能够有效地描述在线客服机器人的自动应答逻辑。通过示例脚本展示了DSL的基本语法和功能，证明了其在描述客服机器人自动应答逻辑方面的有效性。

## 设计与实现

### 数据结构

在本项目中，主要的数据结构包括抽象语法树（AST）节点、符号表、解释器状态等。以下是主要的数据结构定义：

- **Node**：表示抽象语法树中的节点，包含节点类型、名称、值、子节点列表、行号和位置等属性。
- **StackListSymbolTable**：符号表，使用栈列表结构存储符号信息。
- **Interpreter**：解释器类，负责执行脚本，维护全局和运行时符号表，处理输入输出缓冲区等。

### 模块划分

项目主要分为以下几个模块：

- **lexer**：词法分析器，负责将输入字符串分割成标记（token）。
- **parser**：语法分析器，负责将标记序列解析成抽象语法树。
- **interpreter**：解释器，负责执行抽象语法树中的指令。
- **server**：服务器，提供API接口，处理用户请求。
- **user**：用户类，表示使用KRL语言的用户，包含用户ID和解释器实例。

## 功能

项目实现了以下主要功能：

- **词法分析**：将输入字符串分割成标记。
- **语法分析**：将标记序列解析成抽象语法树。
- **脚本执行**：解释器执行抽象语法树中的指令，处理输入输出。
- **API接口**：提供创建用户和运行脚本的API接口。

# 接口

---

## 程序间接口

项目通过API接口提供服务，主要包括以下两个接口：

- **创建用户：** `/create_user`，http post 请求，创建一个新的用户，并返回用户ID和初始输出。
- **运行脚本：** `/run`，http post 请求，根据用户ID和输入字符串运行脚本，并返回输出结果。

## 人机接口

项目提供了一个简单的前端界面，用户可以通过该界面与服务器进行交互，发送输入并接收输出。

当然，也可以通过其它方法向后端发送post请求访问。

## 测试

---

项目使用pytest框架进行测试，测试覆盖了主要功能，包括词法分析、语法分析、脚本执行和API接口。测试代码位于 `tests` 目录下。

## 测试桩


未来会引入AI,但暂未实现，故而先用测试桩代替

file: `src\interpreter\mod.py`

```
1 def generate_text(self, text: str) -> str:
2     '''
3     通过NLP生成用户请求文本
4     '''
5     return text
```

## 自动测试脚本

自动测试脚本使用pytest框架编写，确保项目的主要功能正常工作。



5



36



1

UTF-8

 $\{\}$ 

# 示例脚本

---

以下是几个示例脚本，展示了KRL脚本语言的基本语法和功能：

## 示例1：简单算术运算

```
1 | int x = 3 + 4 * 10;  
2 | print(x);
```

## 示例2：条件语句

```
1 | int x = 5;  
2 | if (x == 5) {  
3 |     print("x is 5");  
4 | } else {  
5 |     print("x is not 5");  
6 | }
```

## 示例3：循环语句

```
1 | int x = 0;  
2 | while (x < 10) {  
3 |     print(x);  
4 |     x = x + 1;  
5 | }
```

## 示例4：函数调用

```
1 | fn factorial(int n) {  
2 |     if (n == 0) {  
3 |         return 1;  
4 |     } else {  
5 |         return n * factorial(n - 1);  
6 |     }  
7 | }  
8 |  
9 | int result = factorial(5);  
10 | print("Factorial of 5 is ${result}");
```

## 示例5：事件处理

```
1 | event "start" {  
2 |     print("welcome to the chatbot!");  
3 |     int choice = get("Enter 1 for weather, 2 for news:");  
4 |     if (choice == 1) {  
5 |         trigger("weather");  
6 |     } else if (choice == 2) {  
7 |         trigger("news");  
8 |     } else {  
9 |         print("Invalid choice");  
10 |     }  
}
```

```
11 }
12
13 event "weather" {
14     print("The weather is sunny.");
15 }
16
17 event "news" {
18     print("Today's news: ...");
19 }
```

## 结论

本项目通过设计和实现一个领域特定脚本语言，展示了DSL在特定领域业务流程定制中的应用。项目实现了词法分析、语法分析、脚本执行和API接口等功能，并通过测试验证了其正确性和稳定性。通过示例脚本展示了KRL脚本语言的基本语法和功能，证明了其在描述客服机器人自动应答逻辑方面的有效性。

## 附录

### 项目概述

- **日期:** 2024-12-20 11:18:03
- **目录:** c:\Users\21756\Desktop\krl
- **文件总数:** 129
- **代码行数:** 14,670
- **注释行数:** 352
- **空行数:** 545
- **总行数:** 15,567

### 语言统计

语言	文件数	代码行数	注释行数	空行数	总行数
JSON	6	10,797	0	8	10,805
Python	30	1,486	59	221	1,766
Vue	32	1,109	47	108	1,264
TypeScript	35	782	190	90	1,062
SCSS	14	276	53	38	367
Markdown	4	143	0	73	216
其他	8	77	3	7	87

### 目录结构

路径	文件数	代码行数	注释行数	空行数	总行数
.	129	14,670	352	545	15,567

路径	文件数	代码行数	注释行数	空行数	总行数
examples	97	13,064	294	278	13,636
src	24	1,370	51	190	1,611
tests	5	90	7	22	119

## 详细信息

- **docs:** 包含项目文档。
- **examples:** 包含前端和服务器的示例代码。
- **src:** 包含项目的主要源代码。
- **tests:** 包含测试代码。

详细信息请参见 [Details](#) 和 [Diff Summary](#)。