



# 北京邮电大学

## 《程序设计实践》

题目：一种领域特定脚本语言的解释器的设计与实现

学 号： 2019211597

姓 名： 卓睿

专 业： 计算机科学与技术

学 院： 计算机学院（国家示范性软件学院）

2021 年 12 月 13 日



# 目录

题目描述 .....	1
描述 .....	1
基本要求.....	1
扩展内容.....	1
脚本设计 .....	2
保留字 .....	2
脚本语法.....	2
Step 的定义 .....	2
字符串的定义 .....	2
语句的定义.....	2
注释的定义.....	3
一些规定 .....	3
开发及测试环境 .....	4
设计思路 .....	5
设计模式与规范 .....	5
数据结构 .....	7
数据类型.....	7
关键变量.....	12
bus 总线变量 .....	12
导出的函数 .....	12
模块划分 .....	14
bus.ts .....	14
interface.ts .....	14

interpreter.ts.....	14
parse.ts .....	14
EditorAndBotton.vue .....	14
Chat.vue.....	14
函数调用关系图 .....	15
测试桩与测试脚本 .....	16
测试样例 .....	17
测试脚本.....	17
测试环境.....	20
测试内容.....	20
输入 .....	20
测试结果.....	20
结果分析.....	21
总结.....	22

# 题目描述

## 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

1. 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。
2. 程序输入输出形式不限，可以简化为纯命令行界面。
3. 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。

## 扩展内容

1. 扩展了脚本文法，允许用户进行变量计算；
2. 实现了语法检查；
3. 实现了脚本错误类型显示，精确到某一行；
4. 实现了多用户同时咨询，实时切换环境；
5. 实现了精良的用户交互体验，直观简洁。

# 脚本设计

## 保留字

Step	Speak	Listen	Branch
Silence	Default	Exit	Culculate

## 脚本语法

### Step 的定义

1. 脚本分为多个 Step;
2. 每一个 Step 的第一行需要以 Step 开头, 其后跟一个空格和自定义的 stepId 用于唯一标识 Step;
3. 每一个 Step 的中间不能出现以 Step 开头的行, 否则会视为两个 Step;

### 字符串的定义

1. 字符串可以分为变量类型和文本类型, 可以使用 + 连接为一个字符串;
2. 变量类型的字符串使用 \$ 符号开头, 后面跟随变量名称;
3. 文本类型的字符串使用双引号包裹, 其中为文本内容;

### 语句的定义

1. 每一个 Step 中拥有多个语句, 每一句为一个动作;
2. 以 Speak 开头的语句作用是在该 Step 被执行时发送消息, 在空格后跟随一个字符串;
3. 以 Listen 开头的语句作用是指定监听的时间, 在空格后跟随一个数字, 单位为秒;
4. 以 Branch 开头的语句的作用是指定分支, 参数为用户的输入和下一个 Step 的 id, 用逗号分隔, 用户的输入为包含匹配;

5. 以 Silence 开头的语句的作用是在用户沉默时跳转，参数为跳转到的 Step 的 id；
6. 以 Default 开头的语句的作用是在以上规则都无法指定下一个 Step 时的默认 Step；
7. 以 Culculate 开头的语句的作用是对变量进行计算，参数以逗号分隔，共三个参数，分别为需要计算的变量，计算完成之后跳转到的 stepId，计算公式，其中计算公式符合 Javascript 语法；
7. 以 Exit 开头的语句的作用是标注执行完该 Step 则结束聊天，没有参数；

## 注释的定义

1. 以 # 开头的行将被忽略；
2. 在每一行中，在字符串外的 # 后的内容将被忽略；

## 一些规定

1. 每一个脚本中至少需要有一个 Step 和一个含有 Exit 语句的 Step；
2. 每一个不含有 Exit 语句的 Step 中都需要有 Default 语句；
3. 每一个 stepId 都需要在脚本中被定义；
4. 同一个 Step 中的每一个 Culculate 都会被执行，stepId 以最后一个 Culculate 的为准；

# 开发及测试环境

1. 系统: Windows 11 专业版 Insider Preview 22509.1011
2. 开发语言: TypeScript, HTML, CSS
3. 开发工具: Visual studio code 1.63.0, @vue/cli 4.5.13, yarn v1.22.17, Git
4. 开发框架: Vue.js 3.0
5. 测试工具: vue-jest@5.0.0
6. 测试环境: Chrome 96.0.4664.93



# 设计思路

本项目采用纯前端的开发模式，用户可以在网页中实时编辑和运行脚本，并且提供聊天框可以直接对话，还可以模拟多用户同时咨询的场景，并且设计了数据结构单独保存每个用户的运行环境和上下文，可以随时恢复，如有需要也便于拆分前后端。

在脚本运行方面，本项目参考 PPT 的实现方式并优化，先将脚本语言解析为语法树，再运行，保证代码的正确性和运行的稳定性。

## 设计模式与规范

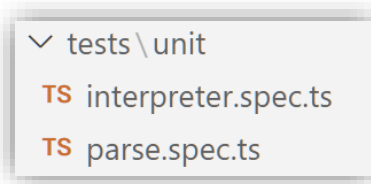
1. 本项目使用拥有类型的 Javascript -- Typescript 进行开发，保证了程序的正确性，保留了 Javascript 敏捷开发的优点，又摒弃了 Javascript 弱类型导致容易错误的问题。

```
export interface BRANCH {  
    /** 跳转分支的回答 */  
    answer: string;  
    /** 要跳转的 stepId */  
    nextStepId: string;  
    /** branch 代码的行号 */  
    line: number;  
}
```

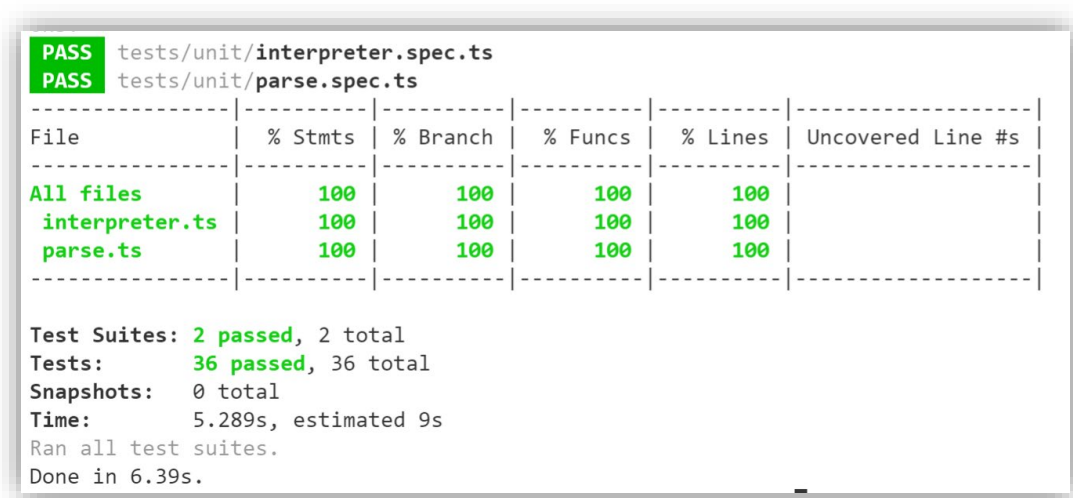
2. 本项目注释详尽，对于关键变量和关键步骤均有描述，且采用 JSDoc 模式的自文档化的注释，可以快速使用 TypeDoc 生成类型和函数注释，方便开发和对接。

```
/**  
 * @desc 分析 code 为语法树  
 * @param code 需要解析的代码  
 * @returns 语法分析树  
 */  
export function parse(code: string): AST {
```

3. 本项目使用 Jest 作为测试框架,使用单元测试的模式对于程序编写了自动化测试脚本,并且对于每一个单元,可以模拟其他单元的正确输出进行测试,即测试桩。



4. 本项目的自动测试脚本覆盖全面,情况考虑完全,共有 2 个测试项目,36 个测试点对 parse 和 interpret 两个关键算法进行测试,语句覆盖率、分支覆盖率、函数覆盖率和行覆盖率均达到了 100%,并且测试通过。



File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	100	100	100	100	
interpreter.ts	100	100	100	100	
parse.ts	100	100	100	100	

Test Suites: 2 passed, 2 total  
Tests: 36 passed, 36 total  
Snapshots: 0 total  
Time: 5.289s, estimated 9s  
Ran all test suites.  
Done in 6.39s.

5. 本项目遵循业界流行的编码规范,在 coding 的代码扫描评分中获得 99.61 的高分,评判工具为:CodeCount、Kunlun-M、Cpd、Eslint、EslintTypeScript、HtmlCs、Lizard、SQ 和 Xcheck。



# 数据结构

## 数据类型

ANSWER	AST	BRANCH	ANSWER
DEFAULT	ENV	HASHTABLE	DEFAULT
LISTEN	MESSAGE	SILENCE	

ANSWER：回答类型

Properties

end

end: *boolean*

Defined in `utils/interface.ts:110`

是否结束聊天

listen

listen: *number*

Defined in `utils/interface.ts:112`

Listen 时间

text

text: *string*

Defined in `utils/interface.ts:108`

回答的内容

7

## AST: 语法树类型

Properties	
<b>HashTable</b>	
○ HashTable: <i>HASHTABLE</i>	
Defined in utils/interface.ts:86	
哈希表	
<b>entry</b>	
○ entry: <i>string</i>	
Defined in utils/interface.ts:88	
入口 Step	
<b>exitStep</b>	
○ exitStep: <i>string[]</i>	
Defined in utils/interface.ts:90	
出口 Step	
<b>vars</b>	
○ vars: <i>VARs</i>	
Defined in utils/interface.ts:92	
变量表	

## BRANCH:

Properties	
<b>answer</b>	
○ answer: <i>string</i>	
Defined in utils/interface.ts:45	
跳转分支的回答	
<b>line</b>	
○ line: <i>number</i>	
Defined in utils/interface.ts:49	
branch 代码的行号	
<b>nextStepId</b>	
○ nextStepId: <i>string</i>	
Defined in utils/interface.ts:47	
要跳转的 stepId	

## DEFAULT: Default 类型

Properties	
<b>args</b>	
○ args: <i>string</i>	
Defined in utils/interface.ts:28	
要跳转的 stepId	
<b>line</b>	
○ line: <i>number</i>	
Defined in utils/interface.ts:30	
default 代码的行号	



## ENV: 运行环境类型

Properties	
<b>curStep</b>	
○ curStep: <i>string</i>	
Defined in utils/interface.ts:100	
当前 Step	
<b>vars</b>	
○ vars: <i>VAR\$</i>	
Defined in utils/interface.ts:102	
变量表	




## HASHTALE: 哈希表类型

Indexable	
🌱 [key: string]: { branch?: <i>BRANCH</i> []; default?: <i>DEFAULT</i> ; line?: <i>number</i> ; listen?: <i>LISTEN</i> ; silence?: <i>SILENCE</i> ; speak?: <i>SPEAK</i> [] }	
▪ Optional <b>branch</b> ?: <i>BRANCH</i> []	branch 类型
▪ Optional <b>default</b> ?: <i>DEFAULT</i>	default 类型
▪ Optional <b>line</b> ?: <i>number</i>	该 Step 的行数
▪ Optional <b>listen</b> ?: <i>LISTEN</i>	listen 类型
▪ Optional <b>silence</b> ?: <i>SILENCE</i>	silence 类型
▪ Optional <b>speak</b> ?: <i>SPEAK</i> []	speak 的列表和行数



## LISTEN: Listen 类型

Properties	
<b>line</b>	
 <code>line: number</code>	
Defined in <code>utils/interface.ts:20</code>	
listen 代码的行号	
<b>time</b>	
 <code>time: number</code>	
Defined in <code>utils/interface.ts:18</code>	
listen 的时间	




## MESSAGE: 消息类型

Properties	
<b>author</b>	
 <code>author: string</code>	
Defined in <code>utils/interface.ts:120</code>	
作者	
<b>data</b>	
 <code>data: { text: string }</code>	
Defined in <code>utils/interface.ts:124</code>	
消息内容	
<b>Type declaration</b>	
<ul style="list-style-type: none"><li>▪ <b>text:</b> <code>string</code> 消息文本</li></ul>	
<b>type</b>	
 <code>type: string</code>	
Defined in <code>utils/interface.ts:122</code>	
消息类型	


## SILENCE: SLIENCE 类型

Properties	
<b>args</b>	
 args: <i>string</i>	
Defined in utils/interface.ts:38	
要跳转的 stepId	
<b>line</b>	
 line: <i>number</i>	
Defined in utils/interface.ts:40	
silence 代码的行号	

## SPEAK: Speak 类型

Properties	
<b>args</b>	
 args: <i>string</i>	
Defined in utils/interface.ts:8	
参数	
<b>line</b>	
 line: <i>number</i>	
Defined in utils/interface.ts:10	
行数	
<b>type</b>	
 type: <i>string</i>	
Defined in utils/interface.ts:6	
speak类型	

## VARS: 变量类型

Indexable	
 [key: <i>string</i> ]: <i>string</i>	

# 关键变量

## bus 总线变量

### Variables

#### bus

Defined in bus.ts:52

#### Type declaration

- **activeCode:** *string*
- **ast:** *AST*
- **defaultCode:** *string*
- **userList:** { [key: string]: { username: string; env: ENV; }; messageList: MESSAGE[]; }; }

# 导出的函数

### parse

🔗 parse(code: *string*): *AST*

Defined in utils/parse.ts:20

desc 分析 code 为语法树

#### Parameters

- **code:** *string*  
需要解析的代码

**Returns** *AST*

语法分析树

### validate

🔗 validate(astToValidate?: *AST*): *void*

Defined in utils/parse.ts:326

验证 ast 语法树的正确性

#### Parameters

- **astToValidate:** *AST* = ast

**Returns** *void*



## getVars

🔗 `getVars(ast: AST): VARS`

Defined in `utils/interpreter.ts:8`

用于获取语法树需要的变量

### Parameters

- **ast:** `AST`  
语法树

**Returns** `VARS`

变量名和变量值

## initEnv

🔗 `initEnv(ast: AST, vars: VARS): ENV`

Defined in `utils/interpreter.ts:18`

### Parameters

- **ast:** `AST`  
语法树
- **vars:** `VARS`  
变量名和变量值

**Returns** `ENV`

返回初始步骤和变量表

## interpret

🔗 `interpret(ast: AST, env: ENV, answerFromUser: string, entry?: boolean, silence?: boolean): ANSWER`

Defined in `utils/interpreter.ts:34`

用于执行脚本

### Parameters

- **ast:** `AST`  
语法树
- **env:** `ENV`  
运行环境
- **answerFromUser:** `string`  
用户的回答
- **entry:** `boolean` = `false`  
入口
- **silence:** `boolean` = `false`  
是否安静

**Returns** `ANSWER`

返回回应

# 模块划分

主要分为六个模块：bus.ts、interface.ts、interpreter.ts、parse.ts、EditorAndBotton.vue、Chat.vue

## bus.ts

总线模块，用于存储一些全局信息，便于在组件间共用。

## interface.ts

是类型定义模块，用于规定和注释自定义的类型，便于编码时使用。

## interpreter.ts

是执行脚本的模块，将语法树和运行环境传进去，获得回复和新的环境。

## parse.ts

是解析脚本的模块，输入脚本，输出语法树。

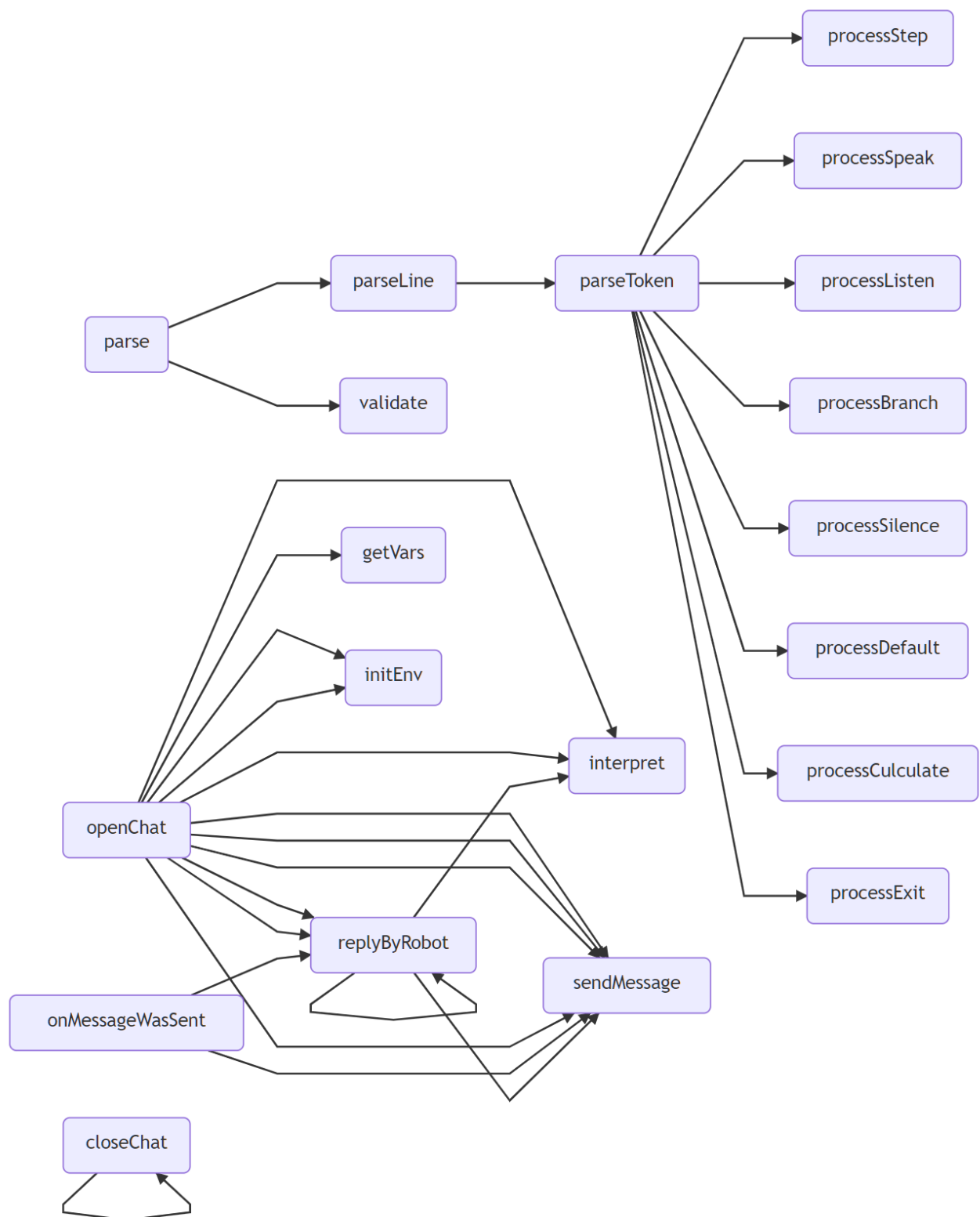
## EditorAndBotton.vue

是编辑器和编辑器相关按钮的模块，可以可视化的编辑脚本和应用。

## Chat.vue

是聊天框的模块，可以让用户在其中直接和应用的脚本的机器人对话。

# 函数调用关系图



# 测试桩与测试脚本

本项目使用 Jest 测试框架，对于 parse 和 interpreter 两个模块进行单元测试，总共有 39 个测试项目。

```
Test Suites: 2 passed, 2 total
Tests:       39 passed, 39 total
Snapshots:   0 total
Time:        9.083s
Ran all test suites.
Done in 10.64s.
```

这 39 个测试项目覆盖了所有的语句、分支、函数和代码行，保证了测试的正确性和完备性。

<b>PASS</b>	tests/unit/interpreter.spec.ts				
<b>PASS</b>	tests/unit/parse.spec.ts (6.536s)				
File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	100	100	100	100	
interpreter.ts	100	100	100	100	
parse.ts	100	100	100	100	

在测试中，Jest 框架承担了测试桩的功能，模拟生成 ast 语法树和用户输入，可以对 parse 和 interpreter 两个模块进行单独测试而不需要两者之间的相互依赖。

Jest 的测试脚本非常自动化，只需要运行 yarn test 即可自动进行测试并输出测试结果。

# 测试样例

## 测试脚本

```
Step welcome
    Speak $name+"您好，请问有什么可以帮您?"
    Listen 5
    Branch "投诉", complainProc
    Branch "账单", billProc
    Branch "查余额", balanceProc
    Branch "查流量", trafficProc
    Branch "充值", chargeProc
    Branch "充流量", buyProc
    Branch "退出", exitProc
    Branch "没有", exitProc
    Branch "买", buyProc
    Silence silenceProc
    Default defaultProc

Step complainProc
    Speak "请问您要投诉什么?"
    Default complainFinishProc

Step complainFinishProc
    Speak "您的意见是我们改进工作的动力，请问还有什么可以帮到您？"
    Listen 5
    Branch "投诉", complainProc
    Branch "账单", billProc
    Branch "查余额", balanceProc
    Branch "查流量", trafficProc
    Branch "充值", chargeProc
    Branch "充流量", buyProc
    Branch "退出", exitProc
    Branch "没有", exitProc
    Branch "买", buyProc
    Silence silenceProc
    Default defaultProc

Step thanks
    Speak "感谢您的来电，再见"
    Exit

Step billProc
    Speak "您的本月账单是" + $amount + "元，请问还有什么可以帮到您？"
    Listen 5
    Branch "投诉", complainProc
    Branch "账单", billProc
    Branch "查余额", balanceProc
    Branch "查流量", trafficProc
    Branch "充值", chargeProc
    Branch "充流量", buyProc
```

```

Branch "退出", exitProc
Branch "没有", exitProc
Branch "买", buyProc
Silence silenceProc
Default defaultProc
Step silenceProc
    Speak "听不清, 请您大声一点可以吗?"
    Branch "投诉", complainProc
    Branch "账单", billProc
    Branch "查余额", balanceProc
    Branch "查流量", trafficProc
    Branch "充值", chargeProc
    Branch "充流量", buyProc
    Branch "退出", exitProc
    Branch "没有", exitProc
    Branch "买", buyProc
    Default defaultProc
Step defaultProc
    Speak "抱歉, 我不明白你的意思"
    Branch "投诉", complainProc
    Branch "账单", billProc
    Branch "查余额", balanceProc
    Branch "查流量", trafficProc
    Branch "充值", chargeProc
    Branch "充流量", buyProc
    Branch "退出", exitProc
    Branch "没有", exitProc
    Branch "买", buyProc
    Silence silenceProc
    Default defaultProc
Step balanceProc
    Speak "您的余额为: " + $balance + "元, 请问还有什么可以帮到您? "
    Listen 5
    Branch "投诉", complainProc
    Branch "账单", billProc
    Branch "查余额", balanceProc
    Branch "查流量", trafficProc
    Branch "充值", chargeProc
    Branch "充流量", buyProc
    Branch "退出", exitProc
    Branch "没有", exitProc
    Branch "买", buyProc
    Silence silenceProc
    Default defaultProc
Step exitProc
    Speak "感谢您的使用, 再见。"
    Exit
Step chargeProc
    Speak "请问您需要充值多少元? 您当前的余额为" + $balance + "元。"
    Culculate $balance, chargeSuccessProc, $balance + INPUT
Step chargeSuccessProc

```

```

Speak "充值成功, 您当前余额为" + $balance + "元, 请问还有什么可以帮到您? "
Listen 5
Branch "投诉", complainProc
Branch "账单", billProc
Branch "查余额", balanceProc
Branch "查流量", trafficProc
Branch "充值", chargeProc
Branch "充流量", buyProc
Branch "退出", exitProc
Branch "没有", exitProc
Branch "买", buyProc
Silence silenceProc
Default defaultProc

Step trafficProc
Speak "您的剩余流量为" + $traffic + "GB, 请问还有什么可以帮到您? "
Listen 5
Branch "投诉", complainProc
Branch "账单", billProc
Branch "查余额", balanceProc
Branch "查流量", trafficProc
Branch "充值", chargeProc
Branch "充流量", buyProc
Branch "退出", exitProc
Branch "没有", exitProc
Branch "买", buyProc
Silence silenceProc
Default defaultProc

Step buyProc
Speak "请问您需要购买多少流量包? 价格为 5 元/GB, 请输入价格。"
Calculate $balance, buySuccessProc, $balance - INPUT
Calculate $traffic, buySuccessProc, $traffic + INPUT / 5

Step buySuccessProc
Speak "流量包购买成功, 余额为" + $balance + "元, 剩余流量为" + $traffic + "GB, 请问还有什么可以帮到您? "
Listen 5
Branch "投诉", complainProc
Branch "账单", billProc
Branch "查余额", balanceProc
Branch "查流量", trafficProc
Branch "充值", chargeProc
Branch "充流量", buyProc
Branch "退出", exitProc
Branch "没有", exitProc
Branch "买", buyProc
Silence silenceProc
Default defaultProc

```

测试环境

\$name	\$amount	\$balance	\$traffic
Jray	23.99	50.0	13.61

测试内容

输入

账单

查余额

查流量

充值

50

( 停顿 10 秒 )

买流量

20

投诉

系统卡顿

没有了

测试结果

其中灰色部分为机器人，蓝色部分为用户

Jray 您好，请问有什么可以帮您？

账单

您的本月账单是 23.99 元，请问还有什么可以帮到您？



查余额

您的余额为：50.0 元，请问还有什么可以帮到您？

查流量

您的剩余流量为 13.61GB，请问还有什么可以帮到您？

充值

请问您需要充值多少元？您当前的余额为 50.0 元。

50

充值成功，您当前余额为 100 元，请问还有什么可以帮到您？

听不清，请您大声一点可以吗？

买流量

请问您需要购买多少流量包？价格为 5 元/GB，请输入价格。

20

流量包购买成功，余额为 80 元，剩余流量为 17.61GB，请问还有什么可以帮到您？

投诉

请问您要投诉什么？

系统卡顿

您的意见是我们改进工作的动力，请问还有什么可以帮到您？

没有了

感谢您的使用，再见。

## 结果分析

经过分析后发现，测试结果和预期一致，运行良好稳定。

# 总结

本项目使用和正式项目一致的开发流程、开发模式和开发规范，注释详细，测试完备。

我在这个项目中也学习到了很多，比如该如何进行单元测试，如何编写测试脚本，如何考虑可能的分支等等。

但是项目中还是有一些问题，比如代码规范没有完全符合更加严格的标准，代码的效率也没有最优化，还是有比较大的进步空间。