# 程序设计与实现

目录

[程序设计与实现 1](#_Toc90207626)

[一、 问题概述 1](#_Toc90207627)

[二、 功能需求分析 1](#_Toc90207628)

[三、 总体方案设计说明 2](#_Toc90207629)

[1 **开发环境** 2](#_Toc90207630)

[2 **程序框图** 2](#_Toc90207631)

[3 **数据结构和模块说明** 3](#_Toc90207632)

[4 **关键模块设计说明** 8](#_Toc90207633)

[四、 脚本示例及测试 13](#_Toc90207634)

[五、 调试及日志 18](#_Toc90207635)

## 问题概述

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。

## 功能需求分析

* 在线客服机器人根据用户的输入作出相应的应答，可以将该过程抽象一个简易自动机，以用户输入作为转移条件，脚本可以转移到不同状态。
* 每个状态应当有一条客服机器人的输出语句，用于应答或引导用户，且该语句应当在每一个状态的一开始进行输出。故定义第一步为Speak
* 每个状态还需要一条接收用户回答的指令，并从用户的回答中抽离出关键信息，作为状态转移条件。故定义Brunch指令匹配用户的回答。
* 除了基本的问答，脚本应当还需要有展示或修改用户数据的功能，故需要定义一些变量的访问和修改的方法。
* 状态机还需要定义基本的开始状态和退出状态，保证程序的正常执行。

## 总体方案设计说明

1. **开发环境**

操作系统：Windows10

开发语言：c++

1. **程序框图**

User模块：读取用户数据

ParserTree模块：读取脚本文件

Parser模块：语法分析

Script模块：构建脚本语法树

当前状态

下一状态

用户反馈

Interpreter模块：解释执行

ParserTree模块：逐行分析

Lexical模块：词法分析

User模块：写回用户数据

1. **数据结构和模块说明**

本软件总共有以下7个模块：

**User模块：**定义并存储用户数据、读取用户数据、写入用户数据。

数据结构：

1. **class** User
2. {
3. **private**:
4. map<string, string> info; //用户信息：<属性，属性对应的值>
5. **public**:
6. User(map<string, string> userInfo)
7. {
8. **this**->info = userInfo;
9. }
11. //获取用户所有信息
12. map<string, string> getAllInfo();
13. //根据参数attribute返回用户信息
14. string getInfo(string attribute);
15. //更新用户数据
16. **void** setInfo(map<string, string> info);
17. };

函数：

1. **void** readUserInfo(string filePath, vector<User>& users);
2. **void** writeUserInfo(string filePath, vector<User>& users);

**Errors模块：**记录错误信息。

数据结构：

1. //错误信息 <错误语句，错误信息，错误所在行数>
2. **class** Error
3. {
4. **public**:
5. string str;
6. string info;
7. **int** pos;
8. Error(string str, string info, **int** pos);
9. ~Error(){}
10. //显示错误信息
11. **void** showInfo();
12. };

**Lexical模块：**词法分析，识别字符串为token流

数据结构：

1. **class** Lexical
2. {
3. **private**:
4. **int** line;       //代码行数
5. **char** ch;        //当前处理字符
6. **int** pos;        //当前处理字符所在位置
7. string sentence;    //存储当前处理的字符串
8. string currentToken;    //当前识别的token
9. vector<token> tokenList;  //存储已识别的token
11. **public**:
12. //识别一行字符串：str，转化为token
13. vector<token> tokenize(string str);
14. //获取下一个待识别字符
15. **char** getChar();
16. //撤回到上一个识别的字符
17. **void**  retract();
18. //处理行注释
19. **void** lineComment();
20. //识别标识符
21. **void** identifier();
22. //识别数字
23. **void** digit();
24. //处理字符串
25. **void** strings();
26. };

**Parser模块：**语法分析，根据设定的文法，检查词法分析得到的token流是否符合文法规范。

数据结构：

1. **class** Parser
2. {
3. **private**:
4. fstream file;   //打开设计的文法文件
5. fstream debug;  //写入文法检测信息
6. **int** line;   //当前处理文件的行数
7. string start;   //起始符
8. set<string> nonTerminal;  //非终结符
9. set<string> terminal; //终结符
10. map<string, vector<string>> production; //产生式
11. map<string, set<string>> first; //非终结符first集合
12. **int** stateNum = 0;   //状态集数目
13. map<**int**, vector<pair<string, set<string>>>> states; //状态集，pair里为生成式和向前看符号集
14. map<**int**, map<string, **int**>> goTo;    //goto函数
15. map<string, map<string, string>> action;    //action函数，即分析表
17. **public**:
18. //逐行解析文法
19. **void** analyse();
20. //将字符串 str 以 spt 分割为多个部分，储存在string容器中
21. **static** vector<string> split(string str, string spt);
22. //将符号串str切割为以非终结符或终结符顺序组成的集合
23. vector<string> cut(string str);
24. //判断是否为非终结符
25. **bool** isNonTerminal(string s);
26. //查找生成式 p 的first集合
27. set<string> findFirst(string p);
28. //构造文法的First集
29. **void** createFirstSet();
30. // 根据初始产生式，生成该状态的项目闭包
31. vector<pair<string, set<string>>> makeClosure(**int** state);
32. //根据传入的状态，生成从该状态可以转移的新状态集，并编写goto函数
33. **void** createNewState(**int** state);
34. //生成action函数
35. **void** createTable();
36. //根据文法产生的状态集，对token流进行文法分析
37. **bool** parserSentence(vector<string> symbol);
38. };

**ParserTree模块：**将文法分析后的数据构建成脚本语法树

数据结构：

1. **class** ParserTree
2. {
3. **private**:
4. Lexical lex;    //调用词法解析器
5. Parser \* parser; //调用文法解析器
6. ifstream file;  //打开编写的脚本文件
7. Script &script; //脚本语法树
8. **int** stepNum;    //Step数量
9. string line;    //当前解析语句
10. **int** row;        //当前解析的行号
12. **public**:
13. //逐行解析脚本文件
14. **void** parseLine();
15. //处理经过词法分析和文法分析后得到的token流，根据关键字信息存入语法树
16. **void** processTokens(vector<string> tokenList);
17. //解析完毕后再粗略的检查语法树是否完备
18. **void** check();
19. //返回脚本语法树
20. Script& getScript();
21. };

**Script模块：**记录脚本语法树

数据结构

1. //状态
2. **class** Step
3. {
4. **private**:
5. stepId id;
6. string name;    //Step名
7. set<string> commands; //该状态包含的指令集
8. set<string> brunchs;  //匹配用户回答的关键词集合
9. map<string, string> transtions;   //根据指令（commands或brunchs）跳转到下一个状态的状态转移表
10. map<actionName, actionParmVector> actionList; //根据动作名获取所要执行动作的参数
12. **public**:
13. //将指令cmd加入指令集commands。
14. **void** addCommand(string cmd);
15. //添加转移函数
16. **void** addTransition(string cmd, string StepName);
17. //添加用户回答分支
18. **void** addBrunch(string brunch);
19. //存储动作和动作参数
20. **void** addAction(actionName name, actionParmVector parm);
21. //返回状态名
22. string getName();
23. //从commands中查找cmd，确认编程人员是否在该Step中添加了cmd指令
24. **bool** isCommandExist(string cmd);
25. //根据指令，查询状态转移表transtions，返回转移目标的Id
26. stepId getTarget(string cmd);
27. //返回用户回答分支，主要用于提示用户，使用户做出有效回答
28. set<string> getBrunchs();
29. //返回状态转移表
30. map<string, string> getTranstions();
31. //获取动作名：name对应的动作参数
32. actionParmVector getActionParms(actionName name);
33. };
35. **class** Script
36. {
37. **private**:
38. vector<Step> steps;   //状态集
39. stepId entry;   //脚本执行入口
40. stepId exit;    //脚本执行出口，可以不唯一，仅用于判断是否设置了出口
41. vector<string> vars;  //变量名
43. **public**:
44. **void** addStep(Step step);
45. **void** setEntry(stepId id);
46. //设置出口，实际上程序可以有多个出口，所以exit仅用于判断是否设置出口。
47. **void** setExit(stepId id);
48. //返回id对应的Step
49. Step getStep(stepId id);
50. stepId getEntry();
51. stepId getExit();
52. vector<string> getVars();
53. vector <Step>& getSteps();
54. };

**Interpreter模块：**解释执行脚本语法树

数据结构：

1. **class** Interpreter
2. {
3. **private**:
4. fstream logFile;        //日志文件
5. Script& script;         //分析得到的脚本语法树
6. stepId currentStepId;   //当前Step（状态、步骤）的Id
7. stepId preStepId;       //上一个Step的Id
8. **bool** interrupt;         //用于使用Back指令时，打断Speak，直接进入Listen
9. string answer;          //存储用户回答
10. map<string, string> userInfo; //当前用户信息
12. **public**:
13. //解析脚本
14. **void** intepret();
15. //表达 expression 语句
16. **void** expressProc(vector<string> expression);
17. string listenProc(vector<string> parms);
18. //比较用户的回答（answer）与参数（parms）对应的用户数据，匹配成功返回真
19. **bool** compareProc(string answer,  vector<string> parms);
20. //parms记录算数表达式，仅限加减
21. **void** computeProc(vector<string> parms);
22. //将用户回答（answer）加入到parms对应的用户数据中
23. **void** addProc(vector<string> parms);
24. //将用户回答从parms对应的用户数据中去除
25. **void** delProc(vector<string> parms);
26. //获取用户回答中的关键词, brunchs为当前状态可识别的关键词列表
27. string getKeywords(string answer, set<string> brunchs);
28. //将临时数据写回用户数据
29. **void** writeBack(User& user);
30. //日志信息
31. **void** debug(string info);
32. };
33. **关键模块设计说明**

程序中语法分析模块为直接引用编译原理编程作业，原理复杂且在本程序中只是简单地用于检查单独每一行的语法错误，故详细设计说明不在此处阐述。

* 1. **Lexical模块**
     1. 识别注释：lineComment()

程序无需处理注释，当识别到 # 后，忽略其之后的内容直到出现换行符为止。其识别自动机如下：

#

\n

others

* + 1. 识别标识符：identifier()

脚本定义的标识符需以字母和下划线开头，下接任意长度的由字母，数字下划线构成的字符串。

标识符识别完毕后，须与定义的脚本关键字库比对，若不是关键字，则储存在符号表中，作为脚本变量。

脚本关键字如下：

"Start","Step","Listen","Default","Speak","Home", "Back",

"Brunch","Compare","Compute","Add","Delete","Wrong","Right","Silence","Exit"

识别标识符的自动机如下：

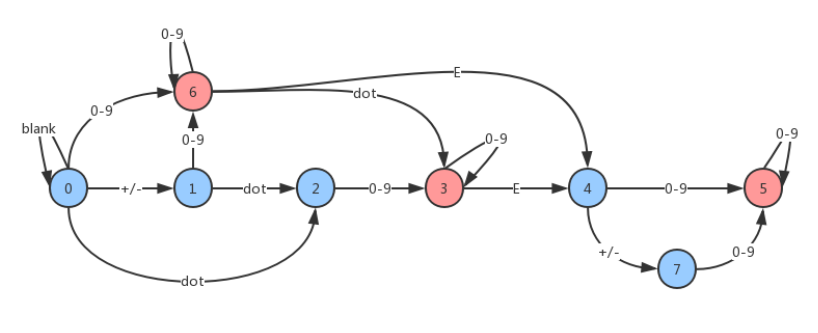
alpha -

others

alpha \_ digit

* + 1. 识别数字：digit()

识别有理数的状态机如下：



* + 1. 识别字符串：strings()

识别字符串常量时需注意，当字符串中含有’\”’，则须将\”添加入token，并且不能将该引号误认为字符串的结束标志。

识别字符串的自动机如下：

“

“

char

\

0nabtvr’”\

* 1. **ParserTree模块**
     1. parserLine()

读取脚本文件中的每一行，传入词法分析程序获得token流，再用语法分析程序检查token流的语法错误。

* + 1. processTokens(vector<string> tokenList)

处理经过词法分析和文法分析后得到的token流，根据关键字信息存入语法树

根据脚本文法定义，tokenList中的第一个必定是脚本关键字，对每个关键字作不同的处理。

1. 若为Start，则创建一个新状态，将当前处理的状态设为该新状态。设置为脚本程序入口。
2. 若为Step，则创建一个新状态，将当前处理的状态设为该新状态 。
3. 若为Exit，则设置该状态为退出状态。
4. 若为状态转移关键字，则将转移条件与转移目标存入状态转移表。
5. 若为其他，则将关键字后的参数列表存入脚本语法树，并记录该关键字，用于解释执行时判断该指令是否存在。
   * 1. check()

此函数用于检查脚本是否设置了入口与出口，以及Step内出现的状态转移目标是否已经定义。记录错误信息。

* 1. **Interpreter模块**

该模块主要功能为读取和解析存储在脚本语法树的内容，如状态转移表、指令和指令参数等。

* + 1. 脚本解释流程图
    2. 指令参数处理

1. 变量：通过语法检查后，&符号后面必定是变量名。

对于变量以及变量的值最简单的处理方法就是使用map容器进行匹配。

程序并没有提供输入变量的接口，故脚本中出现的所有变量名只能为user.txt中已有的变量名。解释器执行时，会先读入用户文件user.txt。使用map容器让用户的属性和值对应。当解释脚本时，通过map容器获得相应变量值。

Start

express

Speak？

N

Y

Listen？

empty？

Default Proc

get answer

Y

N

Silence？

warning

N

Silence Proc

Y

Y

brunch？

Brunch Proc

Y

N

N

Compare？

Compare

Right Proc

Wrong Proc

True

False

Default？

Y

N

Default Proc

warning

Y

N

Default？

Y

Back？

Exit？

Y

Y

N

N

N

Exit

1. Listen指令，要实现至少听num1秒，至多听num2秒。在不开多线程的情况下不易实现，在网上查找到了键盘事件监听的方法，可以大致实现该功能。

用\_kbhit()函数监听键盘事件。当事件发生时，函数返回真，这时再通过getch函数读取字符。事件未发生则返回假，跳过。这样可以避免getch()等系统调用一直阻塞进程。

1. **while** (endTime >= 0)
2. {
3. //至少听 beginTime
4. **if** (\_kbhit() && beginTime <= 0)
5. {
6. **char** ch = \_getch();
7. cout << ch;
8. **if** (ch == 13)   //遇到回车
9. {
10. **break**;
11. }
12. sentence.push\_back(ch);
13. }
14. Sleep(50);
15. endTime--;
16. beginTime--;
17. }
18. Compare指令，用于比对用户answer和程序员设置的参数。

简便起见，多值参数也用string类型表示，每个值后面用’|’进行区分。

Compare有两个功能，若需要比对的参数为单值参数，则直接判断answer和该参数是否相等；若参数为多值参数，则判断answer是否在多值参数中。

1. **if** (userInfo[parms[1]].find("|") != string::npos)
2. {
3. **int** pos = userInfo[parms[1]].find(answer);
4. **if** (pos != userInfo[parms[1]].npos)
5. {
6. //answer应为多值参数中的某一个值，而不能是多值参数中某个值的一部分。
7. **if** (userInfo[parms[1]][pos + answer.size()] == '|')
8. isRight = **true**;
9. }
10. }
11. **else** **if** (userInfo[parms[1]] == answer)
12. isRight = **true**;
13. Compute指令，用于更新用户数据中的数值型数据。数据都以字符串的方式存储，计算时需要使用atoi()函数将字符串转为浮点型。
14. **float** balance = atoi(userInfo[parms[1]].c\_str());
15. **float** deta;
16. **if** (parms[3] == "&") //参数为变量
17. deta = atoi(userInfo[parms[4]].c\_str());
18. **else**  //参数为立即数
19. deta = atoi(parms[3].c\_str());
20. **char** str[100];
21. sprintf\_s(str, 100, "%.2f", balance + deta);
22. userInfo[parms[1]] = str;
23. Add和Delete指令，用于修改用户数据中的多值型数据。

Add只需在数据尾部拼接上answer，Delete需要先查找到answer位置，再通过strng容器的erase函数删除。

## 脚本示例及测试

user.txt

1. phoneNumber
2. id name phoneNumber business remainder balance bill cardNumber password
3. 1 张三 186 语音通话|来电提示| 无限流量|套餐C|套餐D| 50.00 10.00 aaaaaa 111111
4. 2 李四 187 语音通话| 套餐A|套餐B|套餐C|套餐D| 150.00 20.00 bbbbbb 000000

proc.txt

1. Start welcome
2. Speak "尊敬的" + &name + ",你好!"
3. Default home
5. Step home
6. Speak "请问有什么可以帮你?"
7. Listen 1,15
8. Brunch "账单",billProc
9. Brunch "投诉",complainProc
10. Brunch "余额",balanceProc
11. Brunch "业务列表",businessList
12. Brunch "套餐列表",userBusiness
13. Brunch "充值",rechargeProc
14. Brunch "退出",thanksProc
15. Silence silenceProc
16. Default defaultProc
18. Step billProc
19. Speak "你本月账单是" + &bill +"元。请问你还有什么需求？"
20. Back
22. Step complainProc
23. Speak "你的意见是我们改进工作的动力，请问你有什么补充？"
24. Listen 1,15
25. Default thanksProc
27. Step balanceProc
28. Speak "本月余额为:" +&balance +"元。你可以选择充值服务、查询账单服务或返回上级服务菜单。"
29. Listen 1,15
30. Brunch "账单",billProc
31. Brunch "充值",rechargeProc
32. Brunch "上级",home
33. Silence silenceProc
34. Default defaultProc

37. Step silenceProc
38. Speak "抱歉，没有听清楚，可以重复一遍吗？"
39. Back
41. Step defaultProc
42. Speak "非常抱歉，暂时未开放此业务。你可以对我说" + &recommend
43. Back
45. Step thanksProc
46. Speak "感谢你的来电，祝你生活愉快。"
47. Exit
49. Step businessList
50. Speak "你可以对我说" + &recommend
51. Back
53. Step rechargeProc
54. Speak "目前支持充值卡充值，请输入充值卡卡号，或对我说返回上级菜单"
55. Listen 1,15
56. Brunch "上级",home
57. Compare &cardNumber
58. Right inputPsw
59. Wrong invalid
60. Silence silenceProc
61. Default invalid
63. Step invalid
64. Speak "格式错误或输入有误，请重新输入。"
65. Back
67. Step inputPsw
68. Speak "请输入充值卡密码"
69. Listen 1,15
70. Compare &password
71. Right success
72. Wrong wrongPwd
73. Default invalid
75. Step success
76. Speak "充值成功！"
77. Compute &balance , 50
78. Default home
80. Step wrongPwd
81. Speak "密码错误！，请重新输入。"
82. Back
84. Step userBusiness
85. Speak "你办理的套餐有：" +&business +"你可以取消已有套餐、办理新套餐或返回上级菜单"
86. Listen 1,15
87. Brunch "取消",cancelBusiness
88. Brunch "办理", addBusiness
89. Brunch "上级",home
90. Silence silenceProc
91. Default defaultProc
93. Step cancelBusiness
94. Speak "请说出想要取消的套餐。返回上级请说’上级‘"
95. Listen 1,15
96. Brunch "套餐列表",userBusiness
97. Brunch "上级",home
98. Compare &business
99. Right cancelSuccess
100. Wrong cancelWrong
101. Silence silenceProc
102. Default cancelBusiness
104. Step cancelSuccess
105. Speak "取消套餐成功。即月起生效"
106. Delete &business
107. Add &remainder
108. Default home
110. Step cancelWrong
111. Speak "套餐不存在。你可以对我说”套餐列表“获取已有套餐名称"
112. Back
114. Step addBusiness
115. Speak "请说出想要添加的套餐,或者对我说'套餐列表'获取可以添加的套餐，返回上级请说‘上级’"
116. Listen 1,15
117. Brunch "套餐列表",allBusiness
118. Brunch "上级",home
119. Compare &remainder
120. Right addSuccess
121. Wrong addWrong
122. Silence silenceProc
123. Default addBusiness
125. Step addSuccess
126. Speak "添加套餐成功，即月起生效"
127. Add &business
128. Delete &remainder
129. Default home
131. Step addWrong
132. Speak "套餐不存在。"
133. Default addBusiness
135. Step allBusiness
136. Speak "你还可以办理：" + &remainder
137. Back

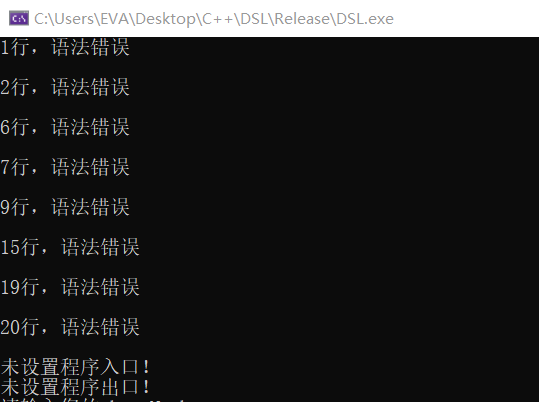
测试结果：

1. 请输入您的phoneNumber:
2. 186
4. 尊敬的张三,您好!
6. 请问有什么可以帮您?
7. 查看业务列表
9. 您可以对我说 充值 套餐列表 投诉 退出 业务列表 余额 账单
10. 我要充值
12. 目前支持充值卡充值，请输入充值卡卡号，或对我说返回上级菜单
13. bbbbbb
15. 格式错误或输入有误，请重新输入。
16. aaaaaa
18. 请输入充值卡密码
19. 222222
21. 密码错误！，请重新输入。
22. 111111
24. 充值成功！
26. 请问有什么可以帮您?
27. 我要干什么
29. 非常抱歉，暂时未开放此业务。您可以对我说 充值 套餐列表 投诉 退出 业务列表 余额 账单
30. 查看套餐列表
32. 您办理的套餐有：语音通话|来电提示|您可以取消已有套餐、办理新套餐或返回上级菜单
33. 返回
35. 非常抱歉，暂时未开放此业务。您可以对我说 办理 取消 上级
36. 我要办理套餐
38. 请说出想要添加的套餐,或者对我说'套餐列表'获取可以添加的套餐，返回上级请说‘上级’
39. A1
41. 套餐不存在。
43. 请说出想要添加的套餐,或者对我说'套餐列表'获取可以添加的套餐，返回上级请说‘上级’
44. 查看可添加的套餐列表
46. 您还可以办理：无限流量|套餐D|套餐C|
47. 套餐C
49. 添加套餐成功，即月起生效
51. 请问有什么可以帮您?
52. 我的套餐列表
54. 您办理的套餐有：语音通话|来电提示|套餐C|您可以取消已有套餐、办理新套餐或返回上级菜单
55. 办理
57. 请说出想要添加的套餐,或者对我说'套餐列表'获取可以添加的套餐，返回上级请说‘上级’
58. 查看套餐列表
60. 您还可以办理：无限流量|套餐D|
61. 返回上级菜单

## 调试及日志

**解释脚本时出现错误**

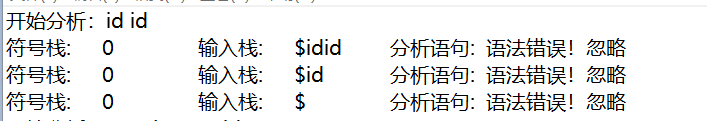
程序会出现以下提示：



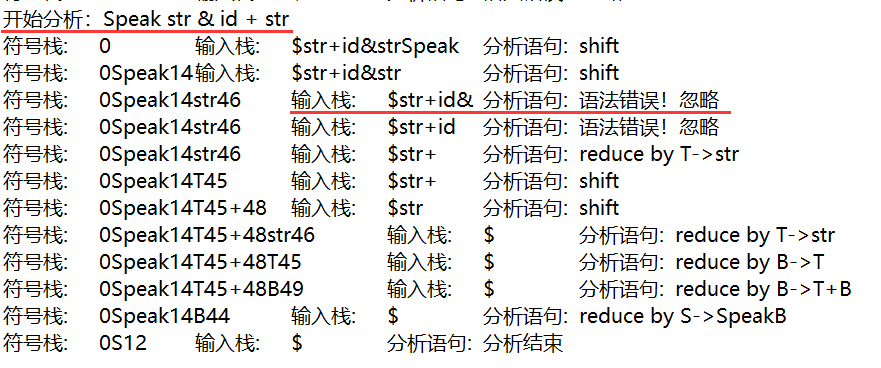
处理方法：

1. 根据行号，可以定位到错误的位置。通过说明文档中脚本设计说明部分可以查找原因。
2. 查看grammar.txt内定义的文法。查看错误语句是否是能由该文法产生。
3. 直接阅读syntaxLog.txt文件。

如下图：



分析的语句是 id id，表明该行只有两个变量名，可能是关键字拼写错误导致误认为变量名。



分析语句为：Speak str & id + str

找到第一处语法错误，此处输入栈为：$str+id& 表示分析到&符号时出现错误。

也即str后面不应该为&符号，而字符串和变量拼接应该用＋号连接。

**执行脚本时不合逻辑**

可以查看interpretLog.txt查看具体执行情况，查找分析错因。

