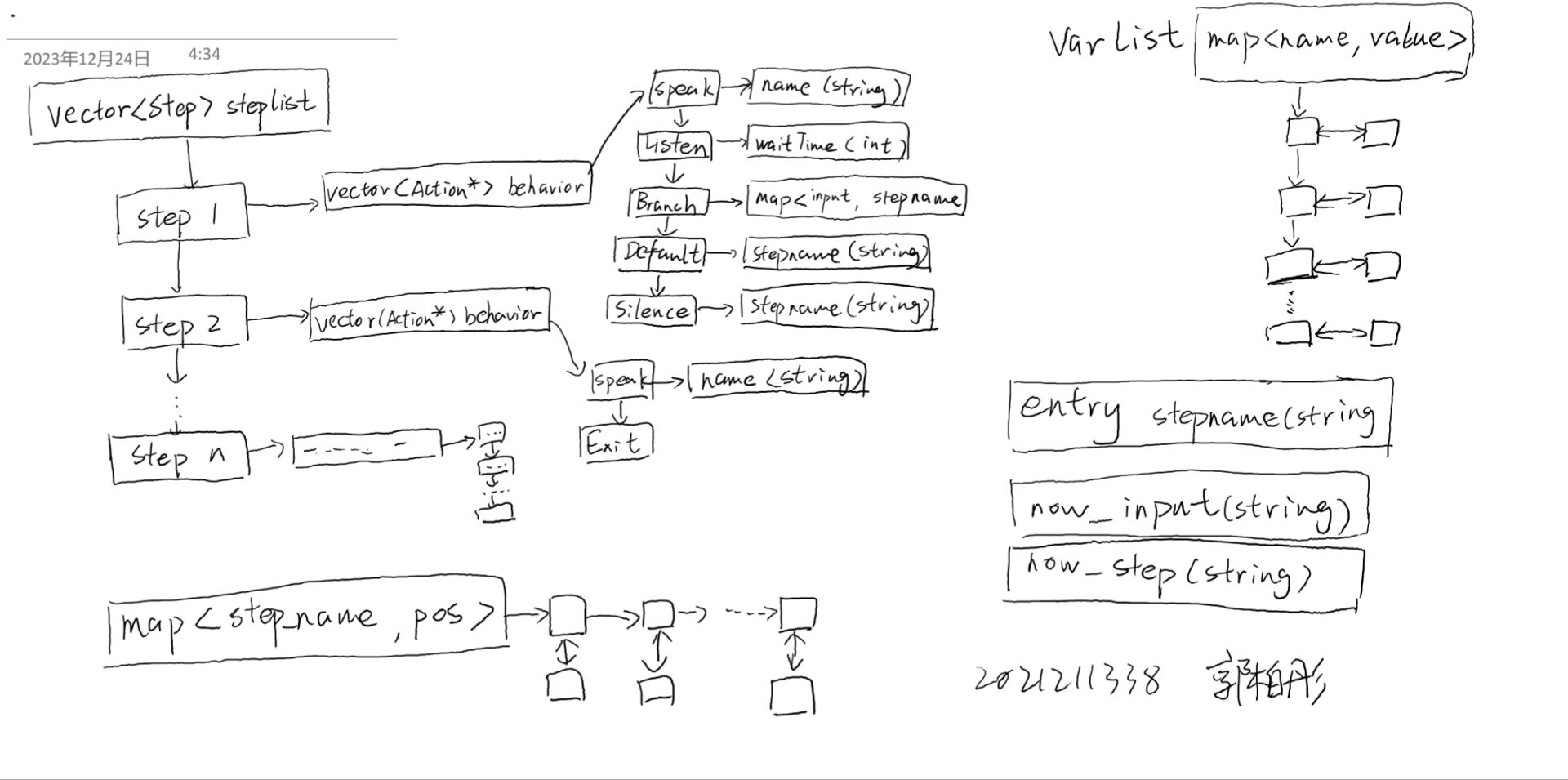
**DSL客服机器人代码设计实现与测试说明**

**310-2021211338-郭柏彤**

**一、代码数据结构设计**

本代码为C++语言代码。主要通过Class类的定义，继承来存储对脚本进行分析得到的结果，具体存储结构为使用C++STL容器vector和map来存储相关的数据。分析结果的存储结构示意图如下：



具体函数定义：

//变量列表

class VarList

{

public:

//从变量名到变量值的映射

map<string, string> var\_list;

};

//操作类，父类

class Action

{

public:

string action\_type;

//虚函数，在继承的子类中详细定义每个操作的详细步骤

virtual int Execute(string& now\_step, string& now\_input, VarList var) = 0;

//用于测试的桩函数

int TEST\_Execute(string& now\_step, string& now\_input, VarList var);

};

//输出

class Speak :public Action

{

public:

int Execute(string& now\_step, string& now\_input, VarList var);

//Speak中要输出的词汇

string words;

};

//根据用户输入进行分支跳转

class Branch : public Action

{

public:

int Execute(string& now\_step, string& now\_input, VarList var);

//从识别内容到跳转的步骤名的映射

map<string, string> jump;

};

//等待用户输入

class Listen : public Action

{

public:

int Execute(string& now\_step, string& now\_input, VarList var);

//预设等待用户输入的时间为10S

int waitTime = 10;

};

//退出客服机器人

class Exit : public Action

{

public:

int Execute(string& now\_step, string& now\_input, VarList var);

};

//等待用户会话超时

class Silence : public Action

{

public:

//下一步跳转的Step名称

int Execute(string& now\_step, string& now\_input, VarList var);

string jumpTo;

};

class Default : public Action

{

public:

//下一步跳转的Step名称

int Execute(string& now\_step, string& now\_input, VarList var);

string jumpTo;

};

//步骤

class Step

{

public:

//Step的名称

string name;

//Step中的各个Action

vector<Action\*> behavior;

};

class Parser

{

public:

~Parser();

//对脚本进行分析的函数，filename为脚本的文件名

bool Parser\_file(const char\* filename);

//测试桩函数，按照预设的一个最简单的脚本进行存储，可保证完全正确，以测试机器人执行部分的准确性

bool TEST\_Parser\_file(const char\* filename);

//客服机器人执行函数

void Run();

private:

//生成对应的Action子类操作，并将操作添加到step\_to\_modify的stepList中

bool SpeakProcess(Step& step\_to\_modify,vector<string> buffer);

bool BranchProcess(Step& step\_to\_modify, vector<string> buffer);

bool ListenProcess(Step& step\_to\_modify, vector<string> buffer);

bool SilenceProcess(Step& step\_to\_modify, vector<string> buffer);

bool ExitProcess(Step& step\_to\_modify, vector<string> buffer);

bool DefaultProcess(Step& step\_to\_modify, vector<string> buffer);

public:

//变量列表

VarList var;

//状态列表

vector<Step> stepList;

//从step名称到step列表中位置的映射

map<string, int> stepPos;

//入口的Step名

string entry;

//当前用户的输入

string now\_input;

//当前的状态

string now\_step;

};

**二、程序执行状态转换定义：**

以下为宏定义，表示各个操作执行之后的状态。

#define ERR 0 //程序运行错误

#define NORMAL 1 //输出后正常执行

#define JUMP 2 //步骤跳转，直接结束当前步骤遍历

#define HEAR 3 //接收到了用户输入，继续正常执行

#define SILENCE 4 //等待用户输入超时，跳过两个操作执行silence操作

#define DEFAULT 5 //结束客服机器人会话

#define EXIT 6 //无法识别用户输入，继续执行下一步的default

执行机器人的主函数为：

//主执行函数

void Parser::Run()

{

//开始，STEP设置为入口STEP

now\_step = entry;

while (true)

{

//每次操作停顿0.5S

Sleep(500);

//顺序执行当前STEP的一系列步骤

int step\_pos = stepPos.find(now\_step)->second;

//cout <<"now\_step name is:" << stepList[step\_pos].name << endl;

int break\_flag = 0;//用于在执行过程中遇到JUMP状态直接退出当前的Step操作遍历

int silence\_flag = 0;//用于遇到超时状态时计算三个步骤，找到对应的Silence操作

Step step\_now\_step = stepList[step\_pos];

for (auto& iter : step\_now\_step.behavior)//遍历当前Step的所有操作

{

if (break\_flag == 1)//JUMP状态，直接退出遍历

{

break;

}

if (silence\_flag > 0)//Silence状态，要跳过两个操作

{

silence\_flag--;

continue;

}

int state = iter->Execute(now\_step, now\_input, var);//State为当前的状态，状态宏定义参见parser.h

//int state = iter->TEST\_Execute(now\_step, now\_input, var);

//cout << state <<endl;

switch (state)

{

case EXIT:

//cout << "EXiT!" << endl;

return;

case ERR:

cout << "客服机器人产生了意外的错误，即将结束程序！" << endl;

return;

case JUMP:

break\_flag = 1;

//cout << "JUMP" << endl;

break;

case HEAR:

//cout << "HEAR" << endl;

break;

case DEFAULT:

//cout << "DEFAULT" << endl;

break;

case NORMAL:

//cout << "NORMAL" << endl;

break;

case SILENCE:

//cout << "SILENCE" << endl;

silence\_flag = 2;//实现跳过两个步骤

break;

default:

cout << "客服机器人产生了意外的错误，即将结束程序！" << endl;

return;

}

}

}

return;

}

**三、程序模块划分：**

该程序分为五个文件：

main.cpp 主函数，实现基础的启动翻译脚本和开始执行模拟客服机器人。

Parser.h 头文件定义，主要包含各种数据结构的定义，类的声明，继承，派生

Parser.cpp 脚本编译器具体过程的实现，详细描述如何识别脚本语言并存储到定义好的数据结构中。

Interpreter.cpp 解释器，按照已经识别并存储好的脚本预设来逐步执行，通过命令行实现人机交互的客服机器人操作。

Test.cpp 定义两个测试桩函数，用于程序的测试功能。

**四、程序的测试和测试桩函数：**

实现两个测试桩函数，分别实现特定状态的返回测试和对预设好一个简单的已经正常存的脚本执行客服机器人操作来检查Run函数的正确性。

测试桩函数如下：

/\*CODE BY 310-2021211338-郭柏彤 程序设计实践 大三上 DSL实现客服机器人\*/

/\*这是一个定义测试所用的桩函数的代码\*/

#include"Parser.h"

#include <Windows.h>

#include <cstdlib>

#include <ctime>

using namespace std;

//测试桩，模拟执行函数正常执行且返回了一定的状态、

int Action::TEST\_Execute(string& now\_step, string& now\_input, VarList var)

{

int x;

srand(time(0));

x = rand()%7;//随机生成状态0-6，状态列表参见parser.h中的宏定义

return x;

}

//测试桩，为了模拟已经成功读入并识别完成一个脚本，来测试是否能正常按照识别好的脚本来运行客服机器人。

bool Parser::TEST\_Parser\_file(const char\* filename)

{

/\*

一个简单的测试用STEP：

Step TEST\_speak\_step

Speak this is a speak step which is used to test program.

Exit

End

\*/

entry = "TEST\_speak\_step";//入口Step

Step new\_step;

new\_step.name = "TEST\_speak\_step";

//Speak操作

Speak\* new\_speak = new Speak;

new\_speak->words = "this is a speak step which is used to test program.";

new\_speak->action\_type = "Speak";

new\_step.behavior.push\_back(new\_speak);

//Exit操作

Exit\* new\_exit = new Exit;

new\_exit->action\_type = "Exit";

new\_step.behavior.push\_back(new\_exit);

//Step加入列表操作

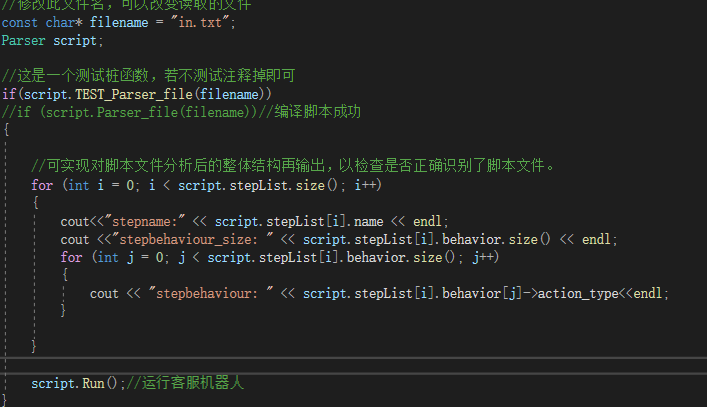
stepList.push\_back(new\_step);

stepPos.insert(pair<string, int>(stepList[0].name, 0));

return true;//返回值一定是成功的

}

如图所示，进行对RUN函数正确性的检查：



将正常执行的函数注释掉，改为执行桩函数

TEST\_Parser\_file(filename)，该函数一定返回一个true并正确将一个简单脚本存储到数据结构中，测试运行结果如下，可以看到RUN函数是正常运行的。

