封面

目录

DSL简介

vlang 是一门用于编写客服机器人的领域专用语言。当你编写一个简单的源文件，就能生成一个可用的客服机器人程序。使用指南（人机接口）见 DSL 语法文档【vlang-Grammar-zh-cn.md】，代码细节详见 doc 文件夹下的 javadoc 文档。

开发者文档

1 集成开发环境

IDE：Jet-Brain IDEA 2020.0.1

语言：java

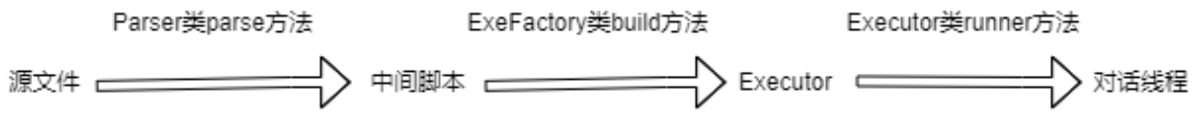
JDK：open-jdk 15.0.2

重要依赖：

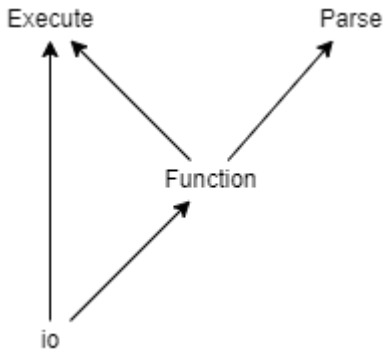
1. gradle-7.1
   （参见 https://github.com/gradle/gradle）
2. javacc 2.0
   （参见 https://javacc.github.io/javacc/）
3. org.json 库 20210307 版
   （参见 https://mvnrepository.com/artifact/org.json/json）
4. org.junit 库 junit 4 （测试使用）
   （参见 https://junit.org/junit4/）

2 流程设计和模块划分

2.1 使用流程设计



2.2 总体划分和依赖关系



代码细节详见 doc 文件夹下的 javadoc 文档

2.2.1 function 模块：函数功能定义，包括语法分析和执行。

函数模块，提供 Listen 、Speak、 Exit 等类，实现包括这些函数在语法分析和执行过程中使用的方法。该模块提供 Function 接口及其实现（上面的 Listen、Speak、Exit 等）。该模块被 parse 模块和 executor 模块调用。该模块中部分函数使用 io 模块的内容，因此依赖于 io 模块。

2.2.2 parse 模块：语法分析和中间脚本生成。

词法和语法分析模块，提供 Parser 接口及其具体实现，实现的主要功能是进行词法和语法分析输出中间脚本。实现中，代码会根据注册在 Registry 类中的 Function 实现调用所有注册函数的语义分析用到的方法。模块依赖于 Fucniton 模块。

2.2.3 execute 模块：根据中间脚本和用户信息运行。

执行器模块，提供 Executor 类和 ExeFactory 类的实现，实现的主要功能是读取 parse 模块输出的中间脚本直接生成执行器，并在有新的用户接入时可以根据用户信息（ClientInfo 类）生成一个 Runable 类的实现，用于生成一个 Thread 实例与用户对话。各对话中用到的语法信息在同一个内存空间。实现中，代码会根据注册在 Registry 类中的 Function 实现调用所有注册函数执行中用到的方法。该模块依赖于 Function 模块及 io 模块。

2.2.4 io 模块：包括输入输出媒体和自然语言处理。

该模块提供输入输出及自然语言处理的功能。为此提供了接口或抽象类 Input、Output、NLP 及其实现。该模块不依赖于上述模块。

2.3 各模块主要包含的文件

1. Function：

   Function.java。

   AskManualService.java、Exit.java、Listen.java、Speak.java。

2. Parse：

   Parser.java。

   JsonParser.java。

   RowJsonParser.jj：

   该文件是基本语法的描述，javacc 源文件，下面的文件由它生成。

   ParseException.java、

   RowJsonParser.java、

   RowJsonParserConstants.java、

   RowJsonParserTokenManager.java、

   SimpleCharStream.java、

   Token、

   TokenMgrError。

3. Execute：

   ClientInfo.java、

   GlobalInfo.java。

   Executor.java。

   Step。

   ExecutorFactory、

   ExeFactoryByJson.java。

4. io：

   InputMedia.java。

   OutputMedia.java。

   NLP.java。

   AnalyzedInputInput.java。

   OutputRowInput.java。

EasyInputMedia.java、

EasyNLP.java、

EasyOutputMedia.java、

EasyRowInputmedia.java。

vlanglOException.java。

3 主要设计

3.1 执行器数据结构

Executor 类的成员：

1. protected final HashMap<String, Step> steps

key 为 Step 名、value 为 Step 的哈希表。

2. protected String entryStep

入口 Step，第一个执行的 Step。

Step 类的成员：

3. private final String name

   Step 名。

4. private final ArrayList<Function> functions

   函数的数组，Step 中依次执行的函数。

3.2 中间脚本

生成的脚本用简单的 JSON 格式模仿存储哈希表和列表。

脚本是一个 JSONObject，用于表示一个从 Step 名到 Step 的哈希表（entryStep）。@entry 标识起始 Step（entryStep）（如图中 59 行），每个 ID 标识一个 Step（name）（如图中 60 行）。

每个 Step 由一个 Function 的列表（functions）表示（如图中 60-78 行）。

每个 Function 调用用一个哈希表表示（如图中 61-67 行），表中 function 标识函数名，param 标识参数列表。

下图是某个源文件 parse 后的中间脚本的一部分。

```
59      "@entry": "welcome",
60      "welcome": [
61          {
62              "param": [
63                  "$name",
64                  "\" Hello, 您好请问有什么可以帮您?\""
65              ],
66              "function": "Speak"
67          },
68          {
69              "goto": {
70                  "$@Default": "thanks",
71                  "投诉": "complainProc",
72                  "$@Silence": "silenceProc",
73                  "账单": "billProc"
74              },
75              "function": "Listen",
76              "time": 10
77          }
78      ]
79  }
```

4 测试

对各函数功能和整体，在工程的 test 目录下作测试。使用到 org.junit 库。

4.1 测试桩：

4.1.1 Word 类：WordTest.java

主要测试其对不同类别符号种类的正确识别（构造方法）。

4.1.2 Listen 类：ListenTest.java

一测试其构造中间脚本对应块功能（buildJson 方法）；

二测试其由中间脚本构造实例的功能（buildByJson 方法）；

三测试其执行功能（exe 方法）。

4.1.3 Speak 类：SpeakTest.java

测试内容同 Listen 类。

4.1.4 RowJsonParser 类：RowJsonParserTest.java

测试其词法分析语法分析结构（parse 方法输出的 JSONObject 实例）。

4.1.5 Executor 类、ExeFactory：ExecutorTest.java

测试 Executor 实例的构造和运行。

（ExeFactory 类的 createBy 方法和 Executor 类的 runner 方法）。

4.2 自动测试脚本：对整体功能

testInputMedia 类是 InputMedia 接口的一个实现。该类为 MainTest.java 提供了从文件输入并且跳过沉默的方法（用空行标识沉默）。这允许对整体功能进行测试时不再被等待沉默时间耽误过多的工程进度。

MainTest.java：对整体进行测试。即读入一个源文件，parse 后读取输出的中间脚本并执行，执行时用读取的输入测试样例文件作为输入。

MainTest 支持连续自动测试多个源文件。

5 接口设计和待拓展内容

5.1 给拓展函数（Function）预留接口

本语言给拓展函数功能预留了开发接口。

要添加一个函数只需要按照以下步骤：

5.2 给接入输入和输出媒体预留接口

本语言给接入输入/输出媒体预留了开发接口。

5.2.1 要接入一个输入媒体只需要按照以下步骤：

1. 实现接口 vlang.io.media.InputMedia

```
   public interface InputMedia <rowInputClass extends RowInput>
   {
      public void gets(String output);
   public void open() throws vlangIOException;
   public void close() throws vlangIOException;
}
```

实现该接口，只需实现其 gets 等方法即可。gets 方法为最长沉默时间为 silenceTime 秒的等待输入函数。返回值为获取到的原始输入，提供给自然语言处理模块使用。

实现该接口应当注意 gets 的返回值类型，也即所用的 rowInputClass，应该与自然语言处理模块相适配。

2. 在 input 中使用该媒体。

可以通过修改 vlang.globalSetting 中注册的 inputMedia 变量的设置或重写 vlang.io.input 类实现。

5.2.2 要接入一个输出媒体只需要按照以下步骤：

1. 实现接口 vlang.io.media.OutputMedia

```
public interface OutputMedia {
      public void puts(String output);
   public void open() throws vlangIOException;
   public void close() throws vlangIOException;
}
```

实现该接口，只需实现其 puts 等方法即可。puts 方法为输出内容字符串形式下为 output 的输出函数。

2. 在 output 中使用该媒体。

   可以通过修改 vlang.globalSetting 中注册的 inputMedia 变量的设置或重写 vlang.io.input 类实现。

5.3 给接入自然语言处理模块预留接口

本语言给自然语言处理模块预留了开发接口。

5.3.1 从输入媒体到自然语言处理模块的中间媒介

1. 实现 vlang.io.rowInput 接口

```
   public interface RowInput {
      public String getString();
      public String getType();
}
```

   实现其 getString 和 getType 方法即可，这两个方法提供给 vlang.io.NLP 接口的实现使用。

2. 该接口也作为 InputMedia 的返回值。实现后应实现一个对应的 InputMedia，其 get 的返回值类型的。这里实现的 rowInput。
3. 该接口实现后还需实现一个 vlang.io.NLP 的实例。其分析方法 analyzeResult 的输入类型应为这里实现的 rowInput。

5.3.2 自然语言处理接口

1. 实现 vlang.io.nlp.NLP 接口

```
    public interface NLP<rowInputClass extends RowInput> {
       public  abstract  AnalyzedInput  analyzeResult(rowInputClass
input, Set<String> targets);
    public void init() throws vlangIOException;
    public void close() throws vlangIOException;
}
```

实现其 annalyzeResult 等方法即可。该方法用于进行分析并返回分析结果。实现时若输入数据对数据有要求，可以实现一个专门的 rowInputClass 来供 NLP 自然语言处理使用。

2. 使用该 NLP 模块。

可以通过修改 vlang.globalSetting 中注册的变量 NLPprocessor 的设置或重写 vlang.io.input 类实现。

5.4 给并发预留接口

本语言给多用户联系客服机器人时需要的并发执行功能预留了开发接口。

由于 Executor 类设计中用 runner 方法给出 Runable 实例的方式发起对话，要实现多用户并发，只需要在根据用户信息和输入输出媒体构造 GlobalInfo 实例后调用 runner 方法并由此产生一个新的线程开始对话。

```
       public Runnable runner(GlobalInfo globalInfo)
```

用法大致如下：

```
    Executor exe = new ExeFactoryByJson().createBy("path");
    . . .
    当有用户接入时：
    在数据库中查找用户信息;
    GlobalInfo newUser= new GlobalInfo(. . .);
    Thread newDialog = new Thread(exe.runner(newUser)));
    newDialog.start();
```

6 补充事项

DSL 文法描述详见语法文档【vlang-Grammar-zh-cn.md】。

使用指南（人机接口）见 DSL 语法文档【vlang-Grammar-zh-cn.md】。

代码细节详见 doc 文件夹下的 javadoc 文档。