Customize Compiler 使用文档

1 项目介绍

本项目实现一个支持自定义词法和自定义部分文法的编译器。项目最终实现了一个支持命令行交互的编译器。用户可以通过指定不同的编译参数使得编译器进行不同的行为。首先，编译生成语法树是整个编译器的核心工作。编译器对源代码文件进行预处理、词法分析、语法分析、语义动作定义，最后生成一棵语法树。在生成语法树的基础上，编译器还支持：执行语法树、将语法树以 json 格式呈现、从语法树生成 java 代码、显示代码覆盖情况等操作。

在实现编译器的过程中，为了达到自定义文法的目的，项目编写了一个语法工厂，当用户改变文法之后，通过运行一次语法工厂即可生成对应的文法解析器。此外，本项目还实现了一些副产品，如：为自定义词法生成 sublime 语法高亮文件、可视化词法正则表达式的自动机。

下面，将依次介绍该本编译器的使用方法。

2 配置文件介绍

配置文件包含两个位于根目录下的文件，分别是 lexer.json 和 production.json。用于自定义词法和自定义文法。自定义词法需要用户掌握一些基本的正则表达式概念，自定义文法则需要用户掌握一些产生式、终结符、非终结符的概念。项目的 Syntax 文件夹包含了预先定义好的三种不同的配置，用户可先行浏览。

2.1 词法定义

项目根目录中包含了一个 lexer.json 文件用来规定词法，lexer.json 的文件格式大致如下所示

```
{
 "language_info": {
  "name": "name",
  "file_extensions": [
   "ncpp"
  ]
 },
 "language_definition": {
  "INT": "int",
```

```
  "REAL": "real",
  "CHAR": "char",
  "IF": "if",
  "ID":
"(_|a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z)(_|a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|
r|s|t|u|v|w|x|y|z|0|1|2|3|4|5|6|7|8|9)*",
  ……
 }
}
```

language_info 包含了生成 sublime 格式的高亮文件所需要的信息。其中 name 为生成的文件名，file_extensions 为需要进行高亮解析的扩展名，用户可以自定义这些信息来规定 sublime 的行为。

language_definition 规定了关键字所对应的正则表达式。如上文中的标识符 ID 对应的正则表达式含义是：小写或下划线开头，然后跟任意数量的下划线、小写或数字。用户首先必须定义素有 lexer.json 中包含的关键字，其次用户也可以定义新的关键字，以供后续的自定义文法时使用。

需要注意的是，编译器在做词法解析时会优先匹配除标识符外的关键字、最后匹配标识符。另外，请不要给不同的关键字定义可能包含相同词素的正则表达式，否则编译器的解析行为将不确定。

2.2 文法定义

production.json 包含了可自定义的文法，一个典型的 production.json 格式如下

```
{
 ……
 "assignment": [
  [
   "ID:mid",
   "ASSIGN",
   "bool:expr",
   "DELIMITER"
  ]
 ],
 "if_else": [
  [
   "IF",
   "BRACKET_LEFT",
   "bool:exp",
   "BRACKET_RIGHT",
   "stmt:s1",
   "ELSE",
   "stmt:s2"
```

```
  ]
 ],……
}
```

每一个文法都是一个长度为 1 的数组。数组的名字如 assignment, if_else 等表示文法的含义。在每个定义的文法中，包含了一系列终结符和非终结符。大写的为非终结符，这些非终结符在 lexer.json 中已经定义，用户也可以使用那些在 lexer.json 中额外定义好的终结符。非终结符包括：type, stmt, bool 分别表示变量类型、语句、表达式。production.json 包含 6 类文法可以自定义，它们的含义及需要满足的条件如下表所示

文法	含义	至少包含的内容	语义动作
declaration	声明语句	type:t 声明的数据类型 ID:mid 声明的标识符名	声明一个类型为 t, 标识符名为 ID 的变量
assignment	赋值语句	ID:mid 赋值的标识符名 bool:expr 需要赋值给 ID 的表达式	将 expr 赋值给变量 mid
if_else	条件语句	bool:exp 条件 stmt:s1 满足条件执行的语句 stmt:s2 不满足条件执行的语句	判断 exp 是否为 true，若为 true 执行 s1, 否则执行 s2
single_if	条件语句	bool:exp 条件 stmt:stmt 满足条件执行的语句	判断 exp 是否为 true，若为 true 执行 stmt，否则跳过
single_while	循环语句	bool:expr 条件 stmt:stmt 满足条件执行的循环	判断 expr 是否为 true，当为 true 时执行 stmt，然后再次判断，直到为 false，跳过 stmt。
do_while	循环语句	stmt:stmt 满足条件执行的循环 bool:expr 条件	先执行一次 stmt，然后行为与 single_while 相同。

表中至少包含的内容是该文法必须包含的非终结符。比如，根据上文 production.json 文件的定义方式，assignment 语句首先是一个终结符 ID，然后是一个终结符 ASSIGN，一个表达式，一个终结符 DELIMITER，那么实际的源代码可以为：

a = 2+3(5-8);*

其中 a 对应 ID，=对应 ASSIGN，2+3*(5-8)对应表达式，; 对应 DELIMITER。

文法的定义相对复杂，用户必须掌握一些产生式的概念，同时还需要对移入/归约冲突有一定的了解以免产生冲突。

2.3 CONST.CUP 规约文件

const.cup 文件规定了最终的规约文件生成的形式。可以在其中指定 production.json 中的终结符优先级等，默认情况下不需要修改，该文件仅建议高级用户尝试使用，如非必要，请不要修改此文件。

3 语法工厂

在定义好 production.json 后，需要运行语法工厂来生成针对定义 production.json 的解析器。语法工厂位于 src/GrammarFactory 中，只需要运行一次该类即可。运行时会输出结果，大致格式为：

```
------- CUP v0.11b 20160615 (GIT 4ac7450) Parser Generation Summary -------
  0 errors and 3 warnings
  39 terminals, 19 non-terminals, and 52 productions declared,
  producing 102 unique parse states.
  3 terminals declared but not used.
  0 non-terminals declared but not used.
  0 productions never reduced.
  0 conflicts detected (0 expected).
  Code written to "Parser.java", and "ParserSym.java".
---------------------------------------------------- (CUP v0.11b 20160615 (GIT 4ac7450))
```

请关注上述输出的 errors 部分，如果输出为 0 errors，则表示运行语法工厂成功。否则，运行工厂失败，查看上面的日志会显示错误的位置，一般产生错误的原因是用户定义的 production.json 产生了冲突，用户需要做相应的修改并在此运行语法工厂，直到运行成功位置。

4 命令行的使用方式

编译器位于 src/Compiler 中，对于不同的编译行为，需要给 Compiler 指定不同的运行参数。

使用参数–help 输出如下使用说明

Usage: compiler [option]

Default:

FILE_NAME : 将 <FILE_NAME> 编译为 <FILE_NAME.ast> 语法树文件

Option:

-exec AST_NAME : 解释执行<AST_NAME> 语法树文件

-json JSON_FILE_NAME AST_NAME : 生成<AST_NAME>语法树 JSON 格式，存在<JSON_FILE_NAME>

-showcoverage SOURCE_NAME : 生成 <SOURCE_NAME> 源代码的代码覆盖率 json 文件

-tojava JAVA_NAME AST_NAME : 生成 < AST_NAME > 语法树的 Java 代码，存在<JAVA_NAME>中

-repl : 启动解释器，进入 读取-求值-打印 循环

-help : 输出命令行使用说明

5 副产品

5.1 生成 SUBLIME 语法高亮配置文件

对于用户自定义的词素为其 Sublime Text 3 的语法高亮配置文件。运行在 Utils.SublimeSyntax.SublimeSyntaxGenerator 类中的 main 方法，会根据当前词法定义 lexer.json 在根目录中生成对应 name.sublime-syntax 文件（ name 为 lexer.json 中定义语言名字的字段）。将该文件放置在：

- OS X: ~/Library/Application Support/Sublime Text 3/Packages/
- Windows: %APPDATA%/Sublime Text 3/Packages/
- Linux: ~/.config/sublime-text-3/Packages/

5.2 正则表达式的自动机可视化

对于用户自定义的词素为其对应的 DFA 生成 Latex 表达式并生成 PDF 可视化。运行在 Lexer.Automata.RegexTree.AutomataVisualization 类中的 main 方法，会在根目录中生成对应 PDF 文件。Mac 中自动调用 PDF_Expert（需安装），Windows 中自动调用默认浏览器打开生成的 PDF 文件。在 Max 和 Windows 中都需要安装 pdflatex 并添加至系统路径中。

5.3 生成抽象语法树对应 JSON 格式

使用编译参数-json <JSON_FILE_NAME> <AST_NAME> 为指定的语法树文件生成对应的 JSON String。