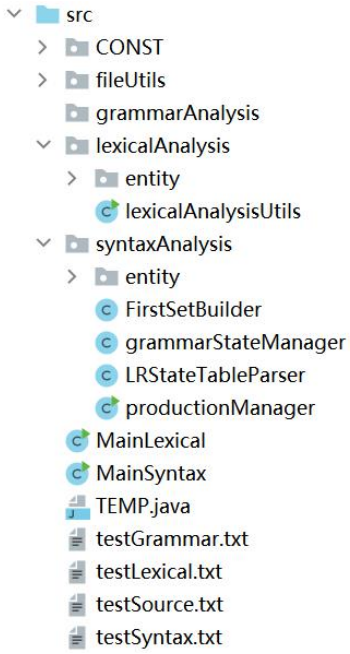
---PanHengyu (njust)

1、Complier 项目文件目录

编译器主要由词法分析、语法分析以及语义分析若干个核心功能组成,由于时间安排等原因，本人只实现了词法分析器以及语法分析器。基于模块化的设计思想，将词法、语法分析器进行模块设计及实现,代码结构如下图所示。



•CONST 文件夹:存放 constPath.java 文件,改文件主要记录 testsource.txt、testLexical.txt、testGrammar.txt 等若干输入文件的路径。

•fileUitils:存放 inputMain.java 以及 inputSystem.java 文件，这两个文件主要实现了文件读取和源程序单词的分割等。

•lexicalAnalysis:①存放了词法分析过程中所需要的实体类:正规文法(FormalGrammar.java)类、NFA 相关类(Nfa.java、NfaFunction.java、NfaNode.java)以及 DFA 相关类(Dfa.java、DfaFunction.java、DfaNode.java)。②lexicalAnalysisUtils.java 文件中实现了从正规文法->NFA->DFA 转化过程。

• syntaxAnalysis:①存放了语法分析过程中所需要的实体类:项目集族(GrammarState.java)类、包含向前搜索符的产生式(Production.java)类以及 2 型文法字符定义(Symbol.java 以及 symbolDefine.java)类。②FirstSetBuilder 类产生 FirstSet 集合的类, grammarStateManager 类是基于 2 型文法生成整个项目集族以及生成 LR(1)分析表类, LRStateTableParser 类是基于 LR(1)分析表判断输入字符

串。

•MainLexical 文件:词法分析器的执行程序所在之处。

•MainSyntax 文件:语法分析器的执行程序所在之处。

•testSource.txt，testLexical.txt，testSyntax.txt:分别存放源程序、正规文法以及 2 型文法。

1.1 关键文件函数注释

词法分析器中的关键逻辑类为 lexicalAnalysis. lexicalAnalysisUtils，其主要函数如下:

函数名	功能实现
tLexical2FGrammar	将 testLexical.txt 的正规文法字符串转化为 FormalGrammar
fGrammar2Nfa	正规文法 FormalGrammar 转化为 NFA
nfa2Dfa	NFA 转化为 DFA
e_closure	转化 DFA 过程中所需的闭包函数
radian_move	转化 DFA 过程中所需的弧转化函数

语法分析器中的关键逻辑类为 syntaxAnalysis.grammarStateManager 和 syntaxAnalysis.GrammarState。其主要函数如下：

函数名	功能实现
syntaxAnalysis.grammarStateManager	
buildTransitionStateMachine	建立 LR(1)项目集族的入口函数
syntaxAnalysis.GrammarState	
makeClosure	建立 LR(1)项目集族的闭包函数
partition	该 LR(1)项目集以向前搜索符右边第一个符号为标识符进行分区
makeTransition	建立 LR(1)项目集族时构造跳转函数

2、程序执行流程

参考编译器设计文档中介绍的 S 语言的相关语法，在 complier 项目 testSource.txt 文件中编写源程序作为输入,倘若点击运行 MainLexical.java 文件中 main 主函数则输出词法分析器的 token 表以及错误信息;倘若点击运行 MainSyntax.java 文件中 main 主函数则输出语法分析器的 LR 分析过程以及判定结果。

2.1、本系统更改语言的方法

考虑到词法分析器和语法分析器的灵活型和适用性,因此对于标识符的识别和常数的识别均采用自动生成 DFA 的识别方法。

因此如果要修改词法分析器,只要修改两个文件即可:①testLexical.txt 中输入识别标识符和常数的正规文法描述, ②lexicalAnalysis.entity.constLexiKeyMap 中修改关键字(key)、限界符(restriction)以及运算符(operator)。修改格式：

```
isIdentier[A]
A->aB
A->cC
B->aB
B->bD
B->cC
B->*
C->aB
C->bD
C->cC
C->*
D->aB
D->bD
D->cC
D->*
```

```
isConst[A]
A->aB
A->bC
A->dD
B->aB
B->dD
B->cF
B->eH
B->bE
C->aE
D->aB
D->bC
E->aE
E->dD
```

```
//初始化关键字
keySet=new HashSet<String>();
keySet.add("int");
keySet.add("float");
keySet.add("boolean");
keySet.add("cin");
keySet.add("cout");
keySet.add("if");
keySet.add("then");
keySet.add("else");
keySet.add("begin");
keySet.add("end");
keySet.add("do");
keySet.add("while");
keySet.add("Const");
```

```
//初始化运算集合
operatorSet=new HashSet<String>();
operatorSet.add("+");
operatorSet.add("-");
operatorSet.add("*");
operatorSet.add("/");
operatorSet.add("%");
operatorSet.add("=");
operatorSet.add("==");
operatorSet.add("!=");
operatorSet.add("<");
operatorSet.add(">");
operatorSet.add("<=");
operatorSet.add(">=");
```

因此如果要修改语法分析器,在设计某种语言 2 型文法之后，修改两个文件即可:①testSyntax.txt 中输入某种语言的 2 型文法，②syntaxAnalysis.entity.symbol-Define 文件中重新定义终结符和非终结符的映射表。**(注:要求设计语言时终结符在语法分析器中的编号大于等于 256，非终结符在语法分析器中的编号小于 256)** 修改格式如下:

```
//设定编号小于256的都是非终结符
//非终结符初始化,
symbolMap.put('S',0);    //<开始符号>
symbolMap.put('A',1);    //<程序>
symbolMap.put('B',2);    //<语句>
symbolMap.put('C',3);    //<常量说明>
symbolMap.put('D',4);    //<变量说明>
symbolMap.put('E',5);    //<常量定义Ⅰ>
symbolMap.put('F',6);    //<常量定义Ⅱ>
symbolMap.put('G',7);    //<标识符Ⅰ>
symbolMap.put('H',8);    //<赋值语句>
symbolMap.put('I',9);    //<表达式>
symbolMap.put('J',10);    //<条件语句>
symbolMap.put('K',11);   //<条件>
symbolMap.put('L',12);   //<当循环语句>
symbolMap.put('M',13);   //<读入语句>
symbolMap.put('N',14);   //<输出语句>
symbolMap.put('O',15);   //<复合语句>
symbolMap.put('P',16);   //<语句Ⅰ>
symbolMap.put('Q',17);   //<表达式串>
```

```
//终结符初始化
symbolMap.put('a',256);tokensMap.put("Const",256);//Const
symbolMap.put('b',257);tokensMap.put("identiter",257);//<标
symbolMap.put('c',258);tokensMap.put("const",258);//<常量>
symbolMap.put('d',259);tokensMap.put("=",259);//=
symbolMap.put('e',260);tokensMap.put("int",260);tokensMap.
symbolMap.put('f',261);tokensMap.put(";",261);  //;
symbolMap.put('g',262);tokensMap.put(",",262);  //,
symbolMap.put('h',263);tokensMap.put("if",263);  //if
symbolMap.put('i',264);tokensMap.put("then",264);  //then
symbolMap.put('j',265);tokensMap.put("else",265);  //else
symbolMap.put('k',266);tokensMap.put("==",266);tokensMap.p
symbolMap.put('l',267);tokensMap.put("while",267);  //whil
symbolMap.put('m',268);tokensMap.put("do",268);  //do
symbolMap.put('n',269);tokensMap.put("cin",269);  //cin
symbolMap.put('o',270);tokensMap.put("(",270);  //(
symbolMap.put('p',271);tokensMap.put(")",271);  //)
symbolMap.put('q',272);tokensMap.put("cout",272);  //cout
symbolMap.put('r',273);tokensMap.put("begin",273);  //begi
symbolMap.put('s',274);tokensMap.put("end",274);  //end
symbolMap.put('t',275);tokensMap.put("+",275);tokensMap.pu
symbolMap.put('u',276);tokensMap.put("*",276);tokensMap.pu
symbolMap.put('v',277);tokensMap.put("{",277);//{
symbolMap.put('w',278);tokensMap.put("}",278);//}
```