## 实验一：使用Antlr4开源工具构建MIDL语言编译器前端补充说明

1. 实验一任务要求**第4条**补充说明。根据MIDL语法规则的文法给出相应的抽象语法树结构，这里的抽象语法树结构是自己进行设计的，因为语法规则文法对应的是语法分析树，需要经过凝练简化成为抽象语法树。下面是一个例子：

*stmt-sequence→statement{ ；statement }*

statement

statement

statement

……

实验内容第4条要求设计文法的抽象语法树并提交设计文档，必要时提供文字说明。

1. 实验一任务要求**第5条**补充说明。针对第4条中设计的抽象语法树，使用Antlr4生成的词法语法分析程序，来构建一个由MIDL源码到抽象语法树的编译器前端。最终呈现是将构建的抽象语法树以树的格式化形式输出到SyntaxOut.txt文件。树的格式化形式可以用缩进来表示父子级关系。例如：

*stmt-sequence*

*statement*

*statement*

*statement*

*...*

1. 实验一任务要求**第6条**补充说明。测试用例需要自己设计构建，测试的目标有两个，① 测试G4中词法，文法是否正确定义。注意antlr4没有报错，只能说明语法没有问题，不代表MIDL规则定义就正确。② 测试你设计的抽象语法树是否正确构建，每个测试用例对应一个SyntaxOut.txt。测试说明写到readme.doc，还有其它需要说明的问题须也写在readme.doc中。
2. 关于如何使用antlr4生成词法，语法分析程序，有两种方法。

首先，确保你在工程引入了org.antlr.v4.runtime 运行时库。

如何构建开发环境 ---> <https://github.com/antlr/antlr4/blob/master/doc/java-target.md>

1. 继承Listener来实现对每类节点的监听，查阅以下链接学习

<https://media.pragprog.com/titles/tpantlr2/listener.pdf>

这种方法的好处是用户无需自己遍历分析树，通过写监听函数即可实现对每类节点的监听。缺点是如果每类节点都有通用的操作，那么用户不得不为每类节点写监听函数，代码比较臃肿。

1. 自己遍历分析树。下面提供了一段Java示例程序供参考学习。这种方法的好处与缺点和第一种刚好相反。自由度更高，自行遍历可以获取到每个节点的属性，并进行一些通用的操作，代码会更精简。缺点是需要自己写递归函数来遍历分析树，对自身算法水平有要求。

|  |
| --- |
| IDLLexer idlL = null;  IDLParser parser = null;  CharStream input = CharStreams.fromString(Util.readFile(idlPath));  idlL = new IDLLexer(input);  CommonTokenStream tokens = new CommonTokenStream(idlL);  parser = new IDLParser(tokens);  ParseTree tree = parser.specification();  if (parser.getNumberOfSyntaxErrors()>0)  throw new SyntaxException(String.format("有 %s 处语法错误，请根据报错信息进行修改后再次尝试 !",parser.getNumberOfSyntaxErrors()));  /\* 渲染所有的stg模板并生成 \*/  context = buildIDLTree(tree,parser);  for (STGroup stg:  stgs) {  targetCodes.add(\_generateTargetCode(context,stg,semanticStg));  }  return targetCodes; |
| public static IDLTreeNode buildIDLTree(ParseTree t,Parser parser) throws NullPointerException{  /\* 错误节点 \*/  if ( t instanceof ErrorNode) {  return null;  }  /\* 终结符节点 \*/  else if ( t instanceof TerminalNode) {  String tokenType = tokensTypeCode.get(((TerminalNode) t).getSymbol().getType());  IDLTreeNode tNode = new IDLTreeNode(tokenType,"T",Trees.getNodeText(t,parser),null,null);  /\* 加入到上下文环境中去 \*/  tNode.setHashCode(tNode.hashCode());  contextMap.put(tNode.hashCode(),tNode);  return tNode;  }  /\* 非终结符节点 \*/  IDLTreeNode root = new IDLTreeNode(Trees.getNodeText(t,parser),"NONT",null,null,null);  root.setHashCode(root.hashCode());  /\* 加入到上下文环境中去 \*/  contextMap.put(root.hashCode(),root);  ArrayList<IDLTreeNode> childs = new ArrayList<IDLTreeNode>();  /\* 对分析树递归 \*/  RuleNode r = (RuleNode)t;  int n = r.getChildCount();  for (int i = 0; i<n; i++) {  ParseTree rc = r.getChild(i);  IDLTreeNode child = buildIDLTree(rc,parser);  childs.add(child);  /\* 为孩子节点设置父亲节点 \*/  child.setPnode(root);  }  /\* 为当前节点设置孩子节点列表 \*/  root.setNodes(childs);  return root;  } |

其中对于tokensTypeCode存的是antlr4生成的IDL.tokens 文件中的token与token id的字典。