Atitit 词法分析法v5 t025 词法分析最佳实践bp

Atitit 词法分析最佳实践bp v4 s529

目录

1.1. 使用状态表比较简单，dfa比较麻烦 状态转移表 1

1.2. 递归的方法来进行词法分析 1

1.3. 双case算法 每个case拆分更具toke 和state组合 1

1.4. 内部分支switch替换if else 2

1.5. error  出错处理的函数，处理出现的词法错误。 2

1.6. Case state命名法规范tokenXXXCase tokenXX\_statXX模式 2

1.7. 单独的db que状态，内部可以容易包含single quo 2

1.8. 单词分类 保留字、标识符、、运算符，，，、分隔符 ，，，常数 2

1.9. Token结果规范 3

1.10. 分词主要流程 3

1.11. 词法分析的第一阶段即扫描器 第二阶段评估器（Evaluator） 3

Lisp分析法，只需要判断(),括号与逗号分隔。。

使用stack来存储零食token非常好

使用异常来代替返回值，null判断

package org.chwin.firefighting.apiserver.QL;  
  
  
import lombok.extern.slf4j.Slf4j;  
import net.sf.jsqlparser.parser.Token;  
import org.jetbrains.annotations.NotNull;  
  
import java.util.ArrayList;  
import java.util.EmptyStackException;  
import java.util.List;  
import java.util.Stack;  
  
@Slf4j  
public class AtiQLParser {  
  
 private final String str;  
 char[] str\_a;  
 int cursor = -1;  
 char cur\_c;  
 Stack stack = new Stack();  
 //List<Token> li\_tokens = new ArrayList<>();  
  
 public AtiQLParser(String s) {  
 str = s;  
 str\_a = s.toCharArray();  
 }  
  
 public static void main(String[] args) {  
  
 // 输出( $(jpql,转换sql结果, 翻页设置是第(3).页, 每页显示(30).条数据);  
 Stack stack = new Stack();  
 String s = " $(from(tab),where( c=5),and(d=xxx),select(a,b,c))";  
  
 char[] a = s.toCharArray();  
 List<Token> li = new AtiQLParser(s).getListToken(s);  
 System.*out*.println(li);  
  
  
 }  
  
 private List<Token> getListToken(String s) {  
 s = s.trim();  
 List<Token> li = new ArrayList<>();  
 Token tk;  
 try {  
 while (true) {  
 tk = getToken();  
 if (tk.image.equals(")"))  
 System.*out*.println("d");  
 if (tk.image.trim().length() == 0)  
 continue;  
 if (tk.image == ",")  
 continue;  
 li.add(tk);  
 }  
  
 } catch (ArrayIndexOutOfBoundsException e) {  
 // li.add(tk);  
 li.add(getTokenTmp(stack.pop()));  
 return li;  
 }  
  
 }  
  
 // only (,)  
 private Token getToken() {  
 cursor++;  
  
 // try{  
 cur\_c = str\_a[cursor];  
  
  
 *log*.info("now char:" + String.*valueOf*(cur\_c));  
 switch (cur\_c) {  
 case '(':  
 return leftBracket\_case();  
 // break;  
  
 case ')':  
 return ritBracket\_case();  
  
 case ',':  
 return comma\_cash();  
  
 default:  
 return Normalchar\_case();  
  
  
 }  
 // cursor++;  
  
 }  
  
 private Token Normalchar\_case() {  
 try {  
 Object obj = stack.pop();  
  
 if (isSplitor(obj)) {  
 Token tk = getTokenTmp(obj);  
  
  
 return tk;  
 }  
 String nowtokenStr = obj.toString() + String.*valueOf*(cur\_c);  
 stack.push(nowtokenStr);  
 return getToken();  
  
  
 } catch (EmptyStackException e) {  
 stack.push(String.*valueOf*(cur\_c));  
 return getToken();  
 }  
  
 }  
  
  
 private Token leftBracket\_case() {  
 Object obj = stack.pop();  
 Token tk = getTokenTmp(obj);  
  
  
 return tk;  
 }  
  
 private Token comma\_cash() {  
 Object obj = stack.pop();  
 Token tk = getTokenTmp(obj);  
  
  
 return tk;  
 }  
  
 private Token ritBracket\_case() {  
 Object obj = stack.pop();  
 Token tk = getTokenTmp(obj);  
  
  
 return tk;  
 }  
  
 @NotNull  
 private Token getTokenTmp(Object obj) {  
 Token tk = new Token();  
 tk.endColumn = cursor;  
 tk.beginColumn = cursor - obj.toString().length();  
 tk.image = obj.toString();  
 stack.push(cur\_c);  
 return tk;  
 }  
  
 private boolean isSplitor(Object obj) {  
 return ("(,)".contains(obj.toString()));  
  
 }  
}