Atitit 词法分析器 版本历史与新特性v8 s529

目录

[1. 版本历史 1](#_Toc16894)

[1.1. V8 1](#_Toc21334)

[1.2. v5 增加对sql单引号的内部支持。可以作为string 1](#_Toc19057)

[1.3. V4 java dsl词法分析 使用循环 1](#_Toc2157)

[1.4. V3 sql的词法分析 1](#_Toc13957)

[1.5. V2 .5 2](#_Toc15942)

[1.6. V2 2](#_Toc21079)

[1.7. V1版 anno 解析器 基于fsm设计模式 2](#_Toc12249)

[1.8. Code /ftpserverati/src/com/attilax/core/TokenizeV8s528.java 2](#_Toc14113)

[2. Ref 7](#_Toc15124)

# 版本历史

## V8

/ftpserverati/src/com/attilax/core/TokenizeV3s528.java

Case state命名法

全面实践最佳实践 Atitit 词法分析最佳实践bp v4 s529

## v5 增加对sql单引号的内部支持。可以作为string

结构调整，使用递归法重构循环发。。放弃循环发。

## V4 java dsl词法分析 使用循环

## V3 sql的词法分析

atitit..sql update语法的词法分析,与语法ast构建 -

## V2 .5

Atitit.antlr实现词法分析 - attilax

## V2

/AtiPlatf\_ee/src/com/attilax/fsm/JavaTokenScannerV2.java

## V1版 anno 解析器 基于fsm设计模式

public class JavaTokenScannerV2 {

## Code /ftpserverati/src/com/attilax/core/TokenizeV8s528.java

package com.attilax.core;

import java.util.List;

import com.alibaba.fastjson.JSON;

import com.attilax.ast.ClassInstanceCreation;

import com.attilax.ast.Expression;

import com.attilax.ast.MethodInvocation;

import com.attilax.ast.SimpleName;

import com.attilax.collection.listBuilder;

import com.attilax.parser.Token;

import com.attilax.str.strService;

import com.google.common.collect.Lists;

public class TokenizeV8s528 {

public static void main(String[] args) {

// new

// com.attilax.core.methodRunner(str:contstuParamVal).methodync(str:haha);

String s = "new com.attilax.core.methodRunner \"string\" contstuParamVal . methDync \"string\" haha";

String[] a = s.split(" ");

List li = listBuilder.$(a).trimElement().delEmptyElement().build();

List<Token> li\_tokens = (List<Token>) new TokenizeV8s528().TokenizeProcess(li);

System.out.println(JSON.toJSONString(li\_tokens, true));

}

List<Token> token\_obj\_list = Lists.newArrayList();

List<String> tokens;

int token\_index = 0;

int paramIdx = 0;

String cur\_stat = "ini";

private Expression Expression;

String cur\_token;

public List<Token> TokenizeProcess(List<String> tokens) {

this.tokens = tokens;

System.out.println("tokenindex" + token\_index);

if (token\_index == 4)

System.out.println("dbg");

if (token\_index >= tokens.size())

return this.token\_obj\_list;

cur\_token = tokens.get(token\_index);

if (cur\_token.equals("."))

System.out.println("d");

switch (cur\_token.trim()) {

case "new":

TokenNew\_cash();

break;

case ".":

TokenPeriod\_case();

break;

default:

tokenNormalchar\_case();

break;

}

// } else if (cur\_token.length() > 0 &&

// cur\_stat.equals("getidnameEndState")) {

// this.cur\_stat = " ParamStart";

token\_index++;

return TokenizeProcess(tokens);

}

private void TokenPeriod\_case() {

switch (cur\_stat) {

case "ParamStart":

Token tk = new Token(cur\_token);

tk.Type = "op";

tk.Text = cur\_token;

token\_obj\_list.add(tk);

this.cur\_stat = "ParamEnd";

break;

}

}

private List<Token> tokenNormalchar\_case() {

switch (cur\_stat) {

case "newopStat":

TokenNormal\_sattNew\_startgetIdname\_ClassOrFunname\_State(cur\_token);

break;

case "ParamStart":

TokenNormal\_statParamstart\_ParamStartStat(cur\_token);

break;

case "ParamEnd":

TokenNormal\_sattNew\_startgetIdname\_ClassOrFunname\_State(cur\_token);

break;

}

return null;

}

private void TokenNew\_cash() {

switch (cur\_stat) {

case "ini":

// if(this.token\_index==0)

tokenNew\_statIni(cur\_token);

}

}

// private void ParamEndStat(String cur\_token) {

// MethodInvocation mi = new MethodInvocation();

// mi.jsonname = "MethodInvocation";

// mi.Exp = this.Expression;

// mi.Name = cur\_token;

// this.Expression = mi;

// this.cur\_stat = "MethodInvocationType";

// }

private void TokenNormal\_statParamstart\_ParamStartStat(String cur\_token) {

Token tk = new Token(cur\_token);

tk.Type = "val";

token\_obj\_list.add(tk);

}

private void TokenNormal\_sattNew\_startgetIdname\_ClassOrFunname\_State(String cur\_token) {

Token tk = new Token(cur\_token);

tk.Type = "id";

token\_obj\_list.add(tk);

// this.stat = "newopStat";

this.cur\_stat = "ParamStart";

}

private List<Token> tokenNew\_statIni(String cur\_token) {

if (cur\_token.equals("invokeMethod")) {

ClassInstanceCreation cic = new ClassInstanceCreation();

// cic.arguments = getArgs(tokens, m\_index);

this.Expression = cic;

this.Expression.jsonname = "ClassInstanceCreation";

this.cur\_stat = "startgetClassname";

//

}

if (cur\_token.equals("new")) {

Token tk = new Token(cur\_token);

tk.Type = "kw";

token\_obj\_list.add(tk);

this.cur\_stat = "newopStat";

//

}

if (cur\_token.equals("invokeStaticMethod")) {

this.Expression = new SimpleName(cur\_token);

this.Expression.jsonname = "ClassInstanceCreation";

this.cur\_stat = "startgetClassname";

}

return TokenizeProcess(this.tokens);

}

private Object getArg(String cur\_token, int paramIdx2, String[] paramtypes) {

String ptype = paramtypes[paramIdx2];

if (ptype.equals("int"))

return Integer.parseInt(cur\_token.toString());

return cur\_token;

}

}

# Ref

Atitit 词法分析的方法attilax总结

《编译原理（第2版）——计算机科学丛书》(（美）阿霍　等著)【简介\_书评\_在线阅读】 - 当当图书.html

编译原理学习--词法分析（1） - Anthony007 - 博客园.html

Java 实现词法分析器（编译原理） - u012577528的专栏 - 博客频道 - CSDN.NET.html

Atitit 词法分析器的设计最佳实践说明attilax总结

Atitit 词法分析最佳实践bp v4 s529