Atitit 语法分析器 版本历史与新特性v6 s529

## V4

/ftpserverati/src/com/attilax/core/AstBuilderV4s529.java

Atitit 语法分析 最佳实践 流程总结 全面实施

双case算法以及规范命名

## V2

下面我们要研究一下递归下降法对文法有什么限制。首先，我们必须要通过超前查看进行分支预测。支持递归下降的文法，必须能通过从左往右超前查看k个字符决定采用哪一个产生式。我们把这样的文法称作LL(k)文法

/AtiPlatf\_cms/src/com/attilax/ast/AstBuilder.java

List<Token> tokens;

int m\_index=0;

public Expressio buildAstV2(List<Token> tokens) {

第二步，如果需要详细了解java的ast结构体系，需要查阅jdt api资料

第三部，构建ast的方法主流使用递归下降法。。至于其他方法较少采用

代码如下

/AtiPlatf\_cms/src/com/attilax/fsm/JavaTokener.java

List<Token> tokens;

int m\_index=0;

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 AstBuilderV4s529 {

public static void main(String[] args) {

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);

com.attilax.ast.Expression ast = new AstBuilderV4s529().buildAstV9s528(li\_tokens);

System.out.println(JSON.toJSONString(ast, true));

}

Token cur\_token;

private Expression Expression;

int paramIdx = 0;

String stat = "ini";

int token\_index = 0;

List<Token> tokens;

public Expression buildAstV9s528(List<Token> 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.Expression;

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

switch (cur\_token.Type.trim()) {

case "kw":

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

TokenTypeKw\_case();

break;

case "id":

tokenTypeId\_case();

break;

case "op":

TokenTypeOp\_case();

break;

case "val":

TokenTypeVal\_case();

break;

default:

// tokenNormalchar\_case();

break;

}

token\_index++;

return buildAstV9s528(tokens);

}

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

String ptype = paramtypes[paramIdx2];

if (ptype.equals("int"))

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

return cur\_token.Text;

}

private boolean isClassName(Token cur\_token) {

// TODO Auto-generated method stub

return (token\_index == 0 || token\_index == 1);

}

private boolean isMethodName(Token cur\_token) {

return !isClassName(cur\_token);

}

private void tokenTypeId\_case() {

switch (this.stat.trim()) {

case "new":

tokenTypeId\_StateNew\_classini(cur\_token);

break;

case "ini":

tokenTypeId\_case\_StateIni\_staticclassini(cur\_token);

break;

default:

tokenTypeId\_StateNOiniNNew\_methodini(cur\_token);

break;

}

}

private void tokenTypeId\_case\_StateIni\_staticclassini(Token cur\_token) {

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

this.Expression.jsonname = "ClassInstanceCreation";

this.stat = "ParamTypeStart";

}

private void tokenTypeId\_StateNew\_classini(Token cur\_token) {

ClassInstanceCreation cic = new ClassInstanceCreation();

cic.name = cur\_token.Text;

this.Expression = cic;

this.Expression.jsonname = "ClassInstanceCreation";

this.stat = "ParamTypeStart";

}

private void tokenTypeId\_StateNOiniNNew\_methodini(Token cur\_token) {

MethodInvocation mi = new MethodInvocation();

mi.jsonname = "MethodInvocation";

mi.Exp = this.Expression;

mi.Name = cur\_token.Text;

this.Expression = mi;

this.stat = "ParamTypeStart";

}

private void TokenTypeKw\_case() {

switch (this.stat.trim()) {

case "ini":

this.stat = "new";

break;

}

}

private void TokenTypeOp\_case() {

if (stat.trim().equals("ParamStart")) {

this.stat = "ParamEnd";

paramIdx = 0;

}

}

private void TokenTypeVal\_case() {

switch (stat.trim()) {

case "ParamTypeStart":

TokenTypeVal\_StateParamTypeStart(cur\_token);

break;

case "ParamStart":

TokenTypeVal\_StateParamStartStat(cur\_token);

break;

case "val":

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

TokenTypeVal\_case();

break;

default:

// tokenNormalchar\_case();

break;

}

}

private void TokenTypeVal\_StateParamStartStat(Token cur\_token) {

// ClassInstanceCreation cic = (ClassInstanceCreation) this.Expression;

Object params = getArg(cur\_token, paramIdx, this.Expression.paramtypes);

this.Expression.arguments.add(params);

// paramIdx++;

}

private void TokenTypeVal\_StateParamTypeStart(Token cur\_token) {

this.stat = "ParamStart";

this.Expression.paramtypes = cur\_token.Text.split(",");

}

}