Java抽象语法树AST，JCTree 分析-程序变量

JCTree简要分析

文章目录

JCinstanceOf
JCLabeledStatement
JCLambda
JCLiteral
JCMemberReference
JCMethodDecl
JCMethodinvocation
JCModifiers
JCNewArray
JCNewClass
JCParens
JCPrimitiveTypeTree
JCReturn
JCSkip
JCStatement
JCSWitch
JCSynchronized
JCThrow
JCTry
JCTypeApply
JCTypeCast
JCTypeIntersection
unknown
JCTypeParameter
JCTypeUnion
JCUnary
JCVariableDecl
JCWhileLoop
JCWildcard

JCAnnotatedType

被注解的泛型：（注解的Target为ElementType.TYPE_USE时可注解泛型）

```
public static class A<T extends @Reality String> {

}
```

JCAnnotatedType @Reality() String
annotations (`List<JCTree.JCAnnotation>`) [0] = (`JCAnnotation`) @Reality()
underlyingType (`JCExpression`) = (`JCIdent`) String

```
protected JCAnnotatedType(List<JCTree.JCAnnotation> var1, JCTree.JCExpression var2) {
    Assert.check(var1 != null && var1.nonEmpty());
    this.annotations = var1;
    this.underlyingType = var2;
}
```

JCAnnotation

注解：@annotationType(args) 例子：@Reality(callSuper = false)

JCAnnotation @Reality(callSuper = false)
annotationType (`JCTree`) = (`JCIdent`) Reality
args (`List<JCExpression>`) = callSuper = false
attribute (`Compound`) = @com.juno.annotations.Reality(callSuper=false)

```
protected JCAnnotation(JCTree.Tag var1, JCTree var2, List<JCTree.JCExpression> var3) {
    this.tag = var1;
    this.annotationType = var2;
    this.args = var3;
}
```

JCArrayAccess

数组访问：a[0] = “123”

index (`JCExpression`) = 0 (`JCLiteral`)
indexed (`JCExpression`) = a (`JCIdent`)

```
protected JCArrayAccess(JCTree.JCExpression var1, JCTree.JCExpression var2) {
    this.indexed = var1;
    this.index = var2;
}
```

JCArrayTypeTree

数组类型：String[] a = new String[2] 中表达式的右边：

elemtype (`JCExpression`) = String (`JCIdent`)

二维数组：String[][] a = new String[1][2]解析为：

`JCArrayTypeTree` String[]
`JCIdent` String

```
protected JCArrayTypeTree(JCTree.JCExpression var1) {
    this.elemtype = var1;
}
```

JCAssert

assert断言：assert cond : detail

```
protected JCAssert(JCTree.JCExpression var1, JCTree.JCExpression var2) {
    this.cond = var1;
    this.detail = var2;
}
```

JCAssign

赋值：lhs = rhs 例如 i = 1

```
protected JCAssign(JCTree.JCExpression var1, JCTree.JCExpression var2) {
    this.lhs = var1; //左表达式
    this.rhs = var2; //右表达式
}
```

JCAssignOp

赋值：lhs opcode rhs 例如 i += 1

```
protected JCAssignOp(JCTree.Tag var1, JCTree var2, JCTree var3, Symbol var4) {
    this.opcode = var1;
    this.lhs = (JCTree.JCExpression)var2;
    this.rhs = (JCTree.JCExpression)var3;
    this.operator = var4;
}
```

赋值符号opcode的可取值：

```
/** Assignment operators, of type Assignop.
 */
BITOR_ASG(BITOR),                // |=
BITXOR_ASG(BITXOR),              // ^=
BITAND_ASG(BITAND),              // &=

SL_ASG(SL),                      // <<=
SR_ASG(SR),                      // >>=
USR_ASG(USR),                    // >>>=
PLUS_ASG(PLUS),                  // +=
MINUS_ASG(MINUS),                // -=
MUL_ASG(MUL),                    // *=
DIV_ASG(DIV),                    // /=
MOD_ASG(MOD),                    // %=
```

JCBinary

将语句分为二叉结构，例如double i = d + i * f 中第一个JCBinary为：

lhs = d
rhs = i * f
operator = null
opcode = PLUS

第二个JCBinary为：

lhs = i
rhs = f
operator = null
opcode = MUL

```
protected JCBinary(JCTree.Tag var1, JCTree.JCExpression var2, JCTree.JCExpression var3, Symbol var4)
    this.opcode = var1;
    this.lhs = var2;
    this.rhs = var3;
    this.operator = var4;
}
```

二元运算符opcode的可取值：

```
/** Binary operators, of type Binary.
 */
OR,                             // ||
AND,                            // &&
BITOR,                          // |
BITXOR,                         // ^
BITAND,                         // &
EQ,                             // ==
NE,                             // !=
LT,                             // <
GT,                             // >
```

```
LE,                                 // <=
GE,                                 // >=
SL,                                 // <<
SR,                                 // >>
USR,                                // >>>
PLUS,                               // +
MINUS,                              // -
MUL,                                // *
DIV,                                // /
MOD,                                // %
```

JCBlock

语句块：{stats} 或 static{stats}（flags = 8L时）

```
protected JCBlock(long var1, List<JCTree.JCStatement> var3) {
    this.stats = var3;
    this.flags = var1;
}
```

JCBreak

break语句：break:label

label 为标签名
target 为跳转目标

```
protected JCBreak(Name var1, JCTree var2) {
    this.label = var1;
    this.target = var2;
}
```

JCCase

case语句：case pat : stats

```
protected JCCase(JCTree.JCExpression var1, List<JCTree.JCStatement> var2) {
    this.pat = var1;
    this.stats = var2;
}
```

JCCatch

catch捕捉异常：catch(param) body

param 定义异常变量 Exception e
body 代码块

```
protected JCCatch(JCTree.JCVariableDecl var1, JCTree.JCBlock var2) {
    this.param = var1;
    this.body = var2;
}
```

JCClassDecl

类的定义：

例子：

```
@Reality
public class MainActivity extends AppCompatActivity {

    private int kk = 1;

    static {
        kk = 2;
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

博客人生

首页

```
        setContentView(R.layout.activity_main);
    }

}
```

mods (`JCModifiers`) = public
name (`Name`) = (`NameImpl`) MainActivity
extending (`JCExpression`) = (`JCIdent`) AppCompatActivity
implementing (`JCExpression`) 实现的接口
defs (`List<JCTree>`) :
defs(1) : JCMethodDecl

```
public <init>() {
    super();
}
```

defs(2) : JCBlock

```
static {
    kk = 2;
}
```

defs(3) : JCVariableDecl

```
private int kk = 1
```

defs(4) : JCMethodDecl

```
@Override()
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    TextView textView = findViewById(R.id.text);
    Random random = new Random();
}
```

sym (`ClassSymbol`) = com.juno.utils.MainActivity

```
protected JCClassDecl(JCTree.JCModifiers var1, Name var2, List<JCTree.JCTypeParameter> var3, JCTree.JC
    this.mods = var1;
    this.name = var2;
```

```
        this.typarams = var3;
        this.extending = var4;
        this.implementing = var5;
        this.defs = var6;
        this.sym = var7;
    }
```

JCCompilationUnit

```
/**
 * Everything in one source file is kept in a TopLevel structure.
 * @param pid The tree representing the package clause.
 * @param sourcefile The source file name.
 * @param defs All definitions in this file (ClassDef,Import,and Ski
 * @param packge The package it belongs to.
 * @param namedImportScope A scope for all named imports.
 * @param starImportScope A scope for all import-on-demands.
 * @param linelMap Line starting positions,defined only if option-g is set.
 * @param docComments A hashtable that stores all documentation comments indexed by the tree nodes the
 * @param endPositions A hashtable that stores ending positions of source ranges indexed by the tree r
 */
public static class JCCompilationUnit extends JCTree implements CompilationUnitTre{
    public List<JCAnnotation>packageAnnotations:
    public JCExpression pid:
    public List<JCTree>defs:
    public JavaFileObject sourcefile:
    public Package Symbol packge:
    public ImportScope namedImportScope:
    public StarImportScope starImportScope:
    public long flags:
    public Position.LinelMap linelMap=null:
    public Map<JCTree,String>docComments=null:
    public Map<JCTree,Integer>endPositions=null:

    protected JCCompilationUnit(List<JCAnnotation>packageAnnotations,
        JCExpression pid,
        List<JCTree>defs,
        JavaFileObject sourcefile,
        Pack ageSymbol packge,
        ImportScope namedImportScope,
        Star ImportScope starImportScope){
            this.packageAnnotations =packageAnnotations:
            this.pid=pid:
            this.defs=defs:
            this.sourcefile=sourcefile:
            this.packge=packge:
            this.namedImportScope=namedImportScope:
            this.starImportScope=starImportScope:
        }
}
```

JCConditional

三目运算符：cond ? truepart : falsepart 例如 a ? 1 : 3

```
protected JCConditional(JCTree.JCExpression var1, JCTree.JCExpression var2, JCTree.JCExpression var3)
    this.cond = var1;
    this.truepart = var2;
    this.falsepart = var3;
}
```

JCContinue

continue跳过一次循环，与break相似：continue label

```
protected JCContinue(Name var1, JCTree var2) {
    this.label = var1;
    this.target = var2;
}
```

JCDoWhileLoop

do-while循环：do{body}while(cond)

```
protected JCDoWhileLoop(JCTree.JCStatement var1, JCTree.JCExpression var2) {
    this.body = var1;
    this.cond = var2;
}
```

JCEnhancedForLoop

增强循环：for(var : expr) body

例如：

```
for (String s : list) {
    print(s);
}
```

JCVariableDecl String s 定义字符串变量s
JCIdent list 遍历的容器
JCBlock { print(s); } 遍历块

```
protected JCEnhancedForLoop(JCTree.JCVariableDecl var1, JCTree.JCExpression var2, JCTree.JCStatement v
    this.var = var1;
    this.expr = var2;
    this.body = var3;
}
```

JCErroneous

Error trees

JCExpression

表达式，赋值、调用方法等都算一个表达式，凡是继承JCExpression都算一个表达式

JCExpressionStatement

内部封装了JCExpression，实际上还是个表达式

```
protected JCExpressionStatement(JCTree.JCExpression var1) {
    this.expr = var1;
}
```

JCFieldAccess

访问父类、其他类的方法、变量时：super.onCreate()

selected (`JCIdent`) = super
name (`NameImpl`) = onCreate

```
protected JCFieldAccess(JCTree.JCExpression var1, Name var2, Symbol var3) {
    this.selected = var1;
    this.name = var2;
    this.sym = var3;
}
```

JCForloop

for循环：

for (init; cond; step) {body}（body为 `JCBlock` ）
for (init; cond; step) body（body为 `JCExpressionStatement` ）

```
protected JCForLoop(List<JCTree.JCStatement> var1, JCTree.JCExpression var2, List<JCTree.JCExpression
    this.init = var1;
    this.cond = var2;
    this.step = var3;
    this.body = var4;
}
```

JCFunctionalExpression

函数式表达式： lambda(JCLambda)和method reference(JCMemberReference)继承于JCFunctionalExpression

JCIdent

出现在表达式中的变量名、调用的方法名以及类名等，例子如下（按顺序）：

Reality 类（注解）
AppCompatActivity 类
String 类
Override 类
Bundle 类
super super和this都是一个JcIdent
savedInstanceState 调用变量的名字
setContentView 方法
R 类
textView 调用变量的名字
findViewById 方法
R类

```
@Reality //JCIdent Reality ClassSymbol com.juno.annotations.Reality
//JCIdent AppCompatActivity ClassSymbol android.support.v7.app.AppCompatActivity
public class MainActivity extends AppCompatActivity  {

    //JCIdent String ClassSymbol java.lang.String
    private static final String TAG = "MainActivity";

    private int kk = 1;
    @Override //JCIdent Override ClassSymbol java.lang.Override
    //JCIdent Bundle ClassSymbol android.os.Bundle
    protected void onCreate(Bundle savedInstanceState) {

        //JCIdent super
        //JCIdent savedInstanceState
        super.onCreate(savedInstanceState);

        //JCIdent setContentView
        //JCIdent R
        setContentView(R.layout.activity_main);

        TextView textView;
        //JCIdent textView
        //JCIdent findViewById
        //JCIdent R
        textView = findViewById(R.id.text);

        int i = 0;
        int j = 1;

        //JCIdent i
        //JCIdent j
        i = j * 3 + 2;

        //JCIdent this
        this.kk = 0;
    }

}
```

其中`TextView textView;`这一句是变量定义语句（`JCVariableDecl`），该语句的`vartype`属性实际上也是个JCIdent：

JCVariableDecl TextView textView
name (`Name`) = textView (`NameImpl`)
vartype (`JCExpression`) = TextView (`JCIdent`)

JCIf

if判断语句：if(condition) {thenpart} else {elsepart}

```
protected JCIf(JCTree.JCExpression var1, JCTree.JCStatement var2, JCTree.JCStatement var3) {
    this.cond = var1;
    this.thenpart = var2;
    this.elsepart = var3;
}
```

JCImport

import语句：

import qulid
static (if staticImport) import qulid

```
protected JCImport(JCTree var1, boolean var2) {
    this.qualid = var1;
    this.staticImport = var2;
}
```

JCinstanceOf

instanceof语句：expr instanceof clazz，例如textView instanceof View：

JCInstanceOf textView instanceof View
clazz (`JCTree`) = (`JCIdent`) View
expr (`JCExpression`) = (`JCIdent`) textView

```
protected JCInstanceOf(JCTree.JCExpression var1, JCTree var2) {
    this.expr = var1;
    this.clazz = var2;
}
```

博客人生

首页

JCLabeledStatement

带有标签的语句：label : body

```
protected JCLabeledStatement(Name var1, JCTree.JCStatement var2) {
    this.label = var1;
    this.body = var2;
}
```

JCLambda

lambda表达式：此处表达式主体只有一句表达式

```
textView.setOnClickListener(v -> v.setVisibility(View.INVISIBLE));
```

JCLambda (v)->v.setVisibility(View.INVISIBLE) （表达式整体）
params (`List<JCTree.JCVariableDecl>`) = (`List`) /*missing*/ v （参数定义列表）
body (`JCTree`) = (`JCMethodInvocation`) v.setVisibility(View.INVISIBLE) （主体）
canCompleteNormally (`Boolean`) = true （未知）
paramKind (`ParameterKind`) = IMPLICIT （参数类型隐形/明确）
getBodyKind() (`BodyKind`) = EXPRESSION （主体是表达式还是块）

当主体变为语句块时：

```
textView.setOnClickListener(v -> {
    v.setVisibility(View.VISIBLE);
});
```

body (`JCTree`) = (`JCBlock`) {v.setVisibility(View.VISIBLE);}
getBodyKind() (`BodyKind`) = STATEMENT

当参数v指明类型为View时：

博客人生

首页

```
textView.setOnClickListener((View v) -> v.setVisibility(View.INVISIBLE));
```

paramKind (`ParameterKind`) = IMPLICIT （参数类型明确）

JCLiteral

表示常量：int a = 1

typetag (`TypeTag`) = INT
value (`Object`) = (`Integer`) 1

```
protected JCLiteral(TypeTag var1, Object var2) {
    this.typetag = var1;
    this.value = var2;
}
```

typetag的可选值有：

```
INT : int
LONG : long
FLOAT : float
DOUBLE : double
CHAR : char
CLASS : String
BOT : null
```

JCMemberReference

lambda表达式中的方法引用：expr::name

例子：new B(“123”).setObjectUtil(ObjectUtil::new);

JCMemberReference ObjectUtil::new

mode (`ReferenceMode`) = NEW
name (`Name`) = (`NameImpl`)
expr (`JCExpression`) = (`JCIdent`) ObjectUtil

例子：new B(“123”).setObjectUtil(this::util);

JCMemberReference this::util
mode (`ReferenceMode`) = INVOKE
name (`Name`) = (`NameImpl`) util
expr (`JCExpression`) = (`JCIdent`) this

```
protected JCMemberReference(ReferenceMode var1, Name var2, JCTree.JCExpression var3, List<JCTree.JCEx
    this.mode = var1;
    this.name = var2;
    this.expr = var3;
    this.typeargs = var4;
}
```

JCMethodDecl

方法的定义：

例子：

```
@Reality
private static <T extends String> int print(@NonNull final String s) throws RuntimeException {
    Log.e(TAG, s);
    return 1;
}
```

mods (`JCModifiers`) = @Reality() private static
name (`Name`) = (`NameImpl`) print
restype (`JCExpression`) = (`JCPrimitiveTypeTree`) int

typarams (`JCTypeParameter`) = T extends String
recvparam = null
params (`List<JCVariableDecl>`) [0] = @NonNull() final String s
thrown (`JCExpression`) = (`JCIdent`) RuntimeException
body (`JCBlock`) = {
Log.e(TAG, s);
return 1;
}
defaultVaule (`JCExpression`) = null
sym (`MethodSymbol`) = print(java.lang.String)

例子：

```
@Reality
@interface B {
    int value() default 1;
}
```

name (`Name`) = (`NameImpl`) value
restype (`JCExpression`) = (`JCPrimitiveTypeTree`) int
typarams (`JCTypeParameter`) 泛型列表为空
recvparam = null
params (`List<JCVariableDecl>`) 参数定义为空
thrown (`JCExpression`) 抛出列表为空
body (`JCBlock`) 方法体为空
defaultVaule (`JCExpression`) = (`JCLiteral`) 1
sym (`MethodSymbol`) = value()

```
protected JCMethodDecl(JCTree.JCModifiers var1, Name var2, JCTree.JCExpression var3, List<JCTree.JCTy
    this.mods = var1;
    this.name = var2;
    this.restype = var3;
    this.typarams = var4;
    this.params = var6;
    this.recvparam = var5;
```

```
        this.thrown = var7;
        this.body = var8;
        this.defaultValue = var9;
        this.sym = var10;
    }
```

JCMethodinvocation

方法执行语句： meth(args)

例子：notNull(before)

meth (`JCExpression`) = (`JCIdent`) notNull
args (`List<JCExpression>`) = (`JCIdent`) before

例子：proxy.invoke(target, method, args)

meth (`JCExpression`) = (`JCFieldAccess`) proxy.invoke
args (`List<JCExpression>`) = (`JCIdent`) target, method, args

```
protected JCMethodInvocation(List<JCTree.JCExpression> var1, JCTree.JCExpression var2, List<JCTree.JCE
        this.typeargs = var1 == null ? List.nil() : var1;
        this.meth = var2;
        this.args = var3;
    }
```

JCModifiers

类、变量、方法等的修饰符和注解：例如：

```
@Reality private static final String TAG = "MainActivity";
```

flags (`long`) = 26L （2L | 8L | 16L)
annotations (`List<JCTree.JCAnnotation>`) = @Reality()

```
protected JCModifiers(long var1, List<JCTree.JCAnnotation> var3) {
    this.flags = var1;
    this.annotations = var3;
}
```

flags的可取值：

```
1L : public
2L : private
4L : protected
8L : static
16L : final
32L : synchronized
64L : volatile
128L : transient
256L : native
1024L : abstract
2048L : strictfp
8589934592L : default
```

方法默认flags不为default，而是0L

JCNewArray

new数组：

例子：String[] s = new String[]{ “123” , “456” };

JCNewArray new String[]{ “123” ， “456” }
elemtype (`JCExpression`) = (`JCIdent`) String

dims (`List<JCTree.JCExpression>`) 空
elems (`List<JCTree.JCExpression>`) [0] = (`JCLiteral`) “123”
elems (`List<JCTree.JCExpression>`) [1] = (`JCLiteral`) “456”

例子：B[][] b = new B[3][4];

JCNewArray new B[3][4]
elemtype (`JCExpression`) = (`JCIdent`) B
dims (`List<JCTree.JCExpression>`) [0] = (`JCLiteral`) 3
dims (`List<JCTree.JCExpression>`) [1] = (`JCLiteral`) 4
elems (`List<JCTree.JCExpression>`) 空

```
protected JCNewArray(JCTree.JCExpression var1, List<JCTree.JCExpression> var2, List<JCTree.JCExpressio
    this.elemtype = var1;
    this.dims = var2;
    this.annotations = List.nil();
    this.dimAnnotations = List.nil();
    this.elems = var3;
}
```

JCNewClass

new一个对象：

例子：new B(“123”)

JCNewClass new B(“123”)
clazz (`JCExpression`) = (`JCTypeApply`) B
args (`List<JCTree.JCExpression>`) [0] = (`JCLiteral`) “123”

```
protected JCNewClass(JCTree.JCExpression var1, List<JCTree.JCExpression> var2, JCTree.JCExpression va
    this.encl = var1;
    this.typeargs = var2 == null ? List.nil() : var2;
    this.clazz = var3;
```

```
        this.args = var4;
        this.def = var5;
    }
```

JCParens

括号：(expr) 存在于if、计算式、synchronized中

```
protected JCParens(JCTree.JCExpression var1) {
    this.expr = var1;
}
```

JCPrimitiveTypeTree

基本类型：

基本类型的赋值：int i = 0 ;

typetag = INT

方法的返回值：void print(String s) {Log.e(TAG, s);}

typetag = VOID

```
protected JCPrimitiveTypeTree(TypeTag var1) {
    this.typetag = var1;
}
```

typetag可选值：

```
INT : int
LONG : long
FLOAT : float
DOUBLE : double
DOOLEAN : boolean
CHAR : char
BYTE : byte
short : short
VOID : void
```

JCReturn

return语句：return expr

```
protected JCReturn(JCTree.JCExpression var1) {
    this.expr = var1;
}
```

JCSkip

空操作，即一个无效的分号 “;”

JCStatement

声明：凡是继承JCStatement都是一个声明，在JCBlock中拿到的都是JCStatement，想在JCBlock中拿到JCExpression就用`JCExpressionStatement`

JCSWitch

switch语句：switch(selector) {cases}

```
protected JCSwitch(JCTree.JCExpression var1, List<JCTree.JCCase> var2) {
    this.selector = var1;
    this.cases = var2;
}
```

JCSynchronized

synchronized同步锁：synchronized(lock){block}

```
protected JCSynchronized(JCTree.JCExpression var1, JCTree.JCBlock var2) {
    this.lock = var1;
    this.body = var2;
}
```

JCThrow

抛出异常：throw expr

```
protected JCThrow(JCTree.JCExpression var1) {
    this.expr = var1;
}
```

JCTry

try块：**try** body catchers **finally** finalizer

```
protected JCTry(List<JCTree> var1, JCTree.JCBlock var2, List<JCTree.JCCatch> var3, JCTree.JCBlock var
    this.body = var2;
    this.catchers = var3;
    this.finalizer = var4;
    this.resources = var1;
}
```

博客人生

首页

resources不知道是什么

JCTypeApply

泛型参数：List list = new ArrayList<>()

对于List list：

clazz (`JCExpression`) = (`JCIdent`) List
arguments (`List<JCTree.JCExpression>`) [0] = (`JCIdent`) String

对于new ArrayList<>()：

clazz (`JCExpression`) = (`JCIdent`) ArrayList
arguments (`List<JCTree.JCExpression>`) = empty

```
protected JCTypeApply(JCTree.JCExpression var1, List<JCTree.JCExpression> var2) {
    this.clazz = var1;
    this.arguments = var2;
}
```

JCTypeCast

类型转换：(clazz)expr

例子：View textView = ((TextView) findViewById(R.id.text));

JCTypeCast (TextView)findViewById(R.id.text)
clazz (`JCTree`) = (`JCIdent`) TextView
expr (`JCExpression`) = (`JCMethodInvocation`) findViewById(R.id.text)

例子：TextView t = (TextView) textView;

JCTypeCast (TextView)textView
clazz (`JCTree`) = (`JCIdent`) TextView
expr (`JCExpression`) = (`JCIdent`) textView

```
protected JCTypeCast(JCTree var1, JCTree.JCExpression var2) {
    this.clazz = var1;
    this.expr = var2;
}
```

JCTypeIntersection

unknown

泛型交叉：

```
public static class A<T extends String & Runnable> {

}
```

```
protected JCTypeIntersection(List<JCTree.JCExpression> var1) {
    this.bounds = var1;
}
```

JCTypeParameter

类的泛型定义：class

JCTypeParameter @Anno() T extends View
name (`Name`) = (`NameImpl`) T
bounds (`List<JCTree.JCExpression>`) [0] = (`JCIdent`) View
JCAnnotation @MainActivity.A()

```
protected JCTypeParameter(Name var1, List<JCTree.JCExpression> var2, List<JCTree.JCAnnotation> var3)
    this.name = var1;
    this.bounds = var2;
    this.annotations = var3;
}
```

JCTypeUnion

catch块中异常的或定义：T1 | T2 | ... Tn

```
try{
    ...
}catch (ClassCastException | ArrayIndexOutOfBoundsException e){
    ...
}
```

JCTypeUnion ClassCastException | ArrayIndexOutOfBoundsException
alternatives (`List<JCTree.JCExpression>`) [0] = (`JCIdent`) ClassCastException
alternatives (`List<JCTree.JCExpression>`) [1] = (`JCIdent`) ArrayIndexOutOfBoundsException

```
protected JCTypeUnion(List<JCTree.JCExpression> var1) {
    this.alternatives = var1;
}
```

JCUnary

一元运算语句：i++中

opcode = POSTINC
arg = i （实际是JCIdent类型）

```
protected JCUnary(JCTree.Tag var1, JCTree.JCExpression var2) {
    this.opcode = var1;
    this.arg = var2;
}

/** Unary operators, of type Unary.
 */
POS,                             // +
NEG,                             // -
NOT,                             // !
COMPL,                           // ~
PREINC,                          // ++ _
PREDEC,                          // -- _
POSTINC,                         // _ ++
POSTDEC,                         // _ --
```

JCVariableDecl

定义变量： mods vartype name = init

例子：final TextView textView = findViewById(R.id.text);

JCVariableDecl final TextView textView = findViewById(R.id.text)
mods (`JCModifiers`) = final
name (`Name`) = (`NameImpl`) textView
vartype (`JCExpression`) = (`JCIdent`) TextView
init (`JCExpression`) = (`JCMethodInvocation`) findViewById(R.id.text)

例子：double i = 1 + 2 * 3;

JCVariableDecl double i = 1 + 2 * 3
name (`Name`) = (`NameImpl`) i
vartype (`JCExpression`) = (`JCPrimitiveTypeTree`) double
init (`JCExpression`) = (`JCBinary`) 1 + 2 * 3

例子：int k = cond ? 0 : 1;

博客人生

首页

JCVariableDecl int k = cond ? 0 : 1
init (`JCExpression`) = (`JCConditional`) cond ? 0 : 1

```
protected JCVariableDecl(JCTree.JCModifiers var1, Name var2, JCTree.JCExpression var3, JCTree.JCExpres
    this.mods = var1;
    this.name = var2;
    this.vartype = var3;
    this.init = var4;
    this.sym = var5;
}
```

JCWhileLoop

while循环：while(cond){body}

为什么body不是Statement List或Block?

```
protected JCWhileLoop(JCTree.JCExpression var1, JCTree.JCStatement var2) {
    this.cond = var1;
    this.body = var2;
}
```

JCWildcard

泛型中的"?"通配符：Class<? super String> c;

JCWildcard ? super String
kind (`TypeBoundKind`) = ? super
inner (`JCTree`) = (`JCIdent`) String

```
protected JCWildcard(JCTree.TypeBoundKind var1, JCTree var2) {
    var1.getClass();
    this.kind = var1;
```

博客人生

首页

```
        this.inner = var2;
    }
```

java AST 源码 分析 JCTree

下一篇　上一篇

相关推荐

ManyMC: m1arm64 macOS 下原生运行 Minecraft 的新选择

Java50道经典编程题：（三）打印水仙花数 ——循环结构的使用

2018 java蓝桥杯校赛题目

Java实现2020-2220年所有回文日

[Java] i++与++i的区别（后缀++与前缀++）

Java实现前缀树

Java后端实习各厂面试编程真题及参考答案（持续更新中...）

蓝桥杯模拟题目——小明种草问题

蓝桥杯 Java试题 D: 分配口罩

蓝桥杯 历届试题 青蛙跳杯子java bfs 实现其中包括java字符串某两个元素交换位置

52-2018 蓝桥杯省赛 B 组模拟赛（一）java

十二届蓝桥杯校内模拟：1到2020有多少个数与2020互质