ACM2023

Semantic

2023. 7. 27 郭一锦

2023

Type System

关于类型需要处理的问题：

基本类型（int、bool等）、自定义类？数组？

Yx

```
package Util;

import java.util.HashMap;

public class Type {
    public boolean isInt = false, isBool = false;
    public HashMap<String, Type> members = null;
}
```

Mx

```
public class Type {

    public String typeName;
    public boolean isClass = false;
    public boolean isArray = false;
    public boolean isConst = false;
    public int dim = 0;

    public ClassDefNode classDecl = null;

    public Type(){

    }
    public Type(String name){
        this.typeName = name;
    }

    public Type(Type other){
        this.typeName = other.typeName;
        this.isArray = other.isArray;
        this.isClass = other.isClass;
        this.dim = other.dim;
    }

    public boolean equals(Type other){
        return this.typeName.equals(other.typeName);
    }

}
```

Type System

```
public abstract class ExprNode extends ASTNode {
    public Type type;
    public entity val;

    public ExprNode(position pos) {
        super(pos);
    }

    public boolean isAssignable() {
        return false;
    }
}
```

e.g.

1. a = b + c;

2. a = b[2][3];

Type System

一种处理方法

```
varDef
   : type varDefUnit (Comma varDefUnit)* Semi;
varDefUnit
   : Identifier (Assign expr)?;
type: typeName ('[' ']')*;
typeName: baseType | Identifier;
baseType: Int | Bool | String;
```

Type System

一种处理方法

```
public class VarDefUnitNode extends ASTNode {

    public TypeNode type;
    public String varName;
    public ExprNode init;

    public VarDefUnitNode(position pos){
        super(pos);
    }
```

关于Scope

Scope表示**作用域**。

semantic 阶段，处理 Scope 的意义是，我们需要保存每个作用域内定义的变量、函数 等等信息，来确保每次在当前作用域或者更内层的作用域中调用某个变量或函数时，我们能够找到它们对应的定义，并判断是否存在语义错误（类型错误、重名等问题）。

关于Scope

```
public class Scope {

    public HashMap<String, VarDefUnitNode> variableMembers = new HashMap<>();
    public HashMap<String, FuncDefNode> functionMembers = new HashMap<>();
    public HashMap<String, entity> entities = new HashMap<>();
    public Scope parentScope;
    public boolean hasReturn = false;

    public ClassDefNode InClass = null;

    public Scope(Scope parentScope){
        this.parentScope = parentScope;
    }
```

关于Scope

```
@Override
public void visit(blockStmtNode it) {
    if (!it.stmts.isEmpty()) {
        currentScope = new Scope(currentScope);
        for (StmtNode stmt : it.stmts) stmt.accept(this);
        currentScope = currentScope.parentScope();
    }
}
```

Next step:

SymbolCollector
and
SemanticChecker

基本方法

```
public class SymbolCollector implements ASTVisitor {
    private globalScope gScope;
    public SymbolCollector(globalScope gScope) {
        this.gScope = gScope;
    }
    @Override
    public void visit(RootNode it) {
        it.strDefs.forEach(sd -> sd.accept(this));
    }

    @Override public void visit(structDefNode it) {
        Type struct = new Type();
        struct.members = new HashMap<>();
        it.varDefs.forEach(vd -> vd.accept(this));
        gScope.addType(it.name, struct, it.pos);
    }
```

基本方法

```java
public class SemanticChecker implements ASTVisitor {
    private Scope currentScope;
    private globalScope gScope;
    private Type currentStruct = null;

    public SemanticChecker(globalScope gScope) {
        currentScope = this.gScope = gScope;
    }

    @Override
    public void visit(RootNode it) {
        it.strDefs.forEach(sd -> sd.accept(this));
        // we SHOULD check struct definitions first
        it.fn.accept(this);
    }
```

SymbolCollector

```
class A {
    int x;
    int y;
};

class B{
    A size;
    int value;
};
```

Symbol Collector 类是一个全局作用域内的收集。

Mx 会在全局自定义类，自定义类也会出现在各个函数和变量定义中（包括出现在其他的类定义中，如图），所以在遍历 AST 进行semantic check之前，我们需要先收集并保存所有的自定义类（int, bool等基本类型也需要保存）。

当然，收集过程中也需要做一些语义检查，比如重名等。

SymbolCollector

注意内建方法——

在初始化globalScope的时候需要考虑的问题，包括无需定义的基本类型（int，bool，string等）、函数（如图）、字符串的内建方法等等。

字符串的处理？

9.2. 内建函数

以下内建函数不需要定义或声明就直接可以使用：

- 函数：`void print(string str);`
  - 作用：向标准输出流中输出字符串 `str`
- 函数：`void println(string str);`
  - 作用：向标准输出流中输出字符串 `str`，并在字符串末输出一个换行符
- 函数：`void printInt(int n);`
  - 作用：向标准输出流中输出数字 `n`
- 函数：`void printlnInt(int n);`
  - 作用：向标准输出流中输出数字 `n`，并且在数字末输出一个换行符
- 函数：`string getString();`
  - 作用：从标准输入流中读取一行字符串并返回
- 函数：`int getInt();`
  - 作用：从标准输入流中读取一个整数，读到空格、回车符、制表符处停止，返回该整数
- 函数：`string toString(int i);`
  - 作用：把整数 `i` 转换为字符串并返回

Semantic Checker

遍历 AST 所有的节点，并对每个节点可能出现的错误进行判断。

```
@Override
public void visit(ifStmtNode it) {
    it.condition.accept(this);
    if (!it.condition.type.isBool)
        throw new semanticError("Semantic Error: type not match. It should be bool",
                it.condition.pos);
    it.thenStmt.accept(this);
    if (it.elseStmt != null) it.elseStmt.accept(this);
}
```

类型检查

- **类型推断**

  根据上下文等信息确定类型

- **类型匹配**

  函数调用的参数？

  操作符对应的类型？

  函数返回值的类型？

  数组类型？

  右值问题？

作用域

变量和函数声明冲突？

控制流语句

break和continue的用法？

Q × A