ACM2023

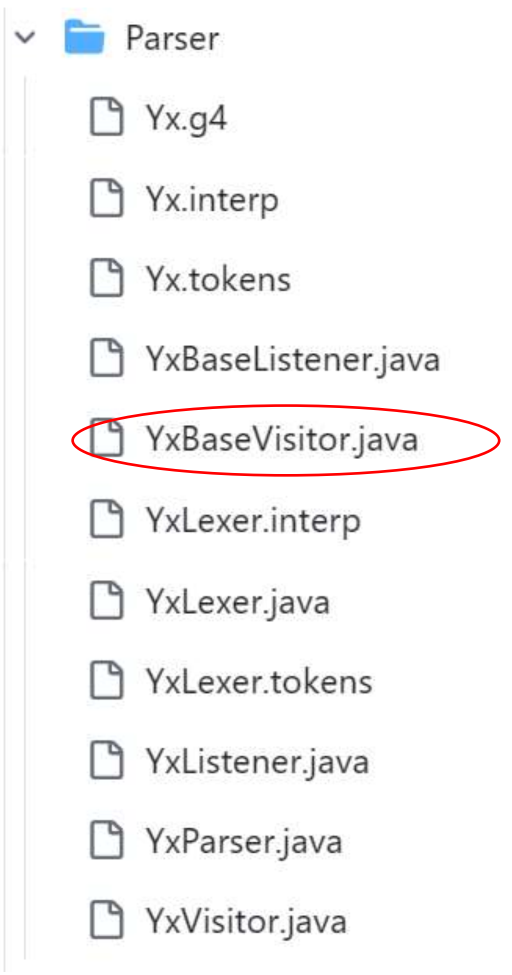
关于AST

2023. 7. 20 郭一锦

Yx： https://github.com/ZYHowell/Yx

20
23

antlr4生成文件



Parser
- Yx.g4
- Yx.interp
- Yx.tokens
- YxBaseListener.java
- YxBaseVisitor.java
- YxLexer.interp
- YxLexer.java
- YxLexer.tokens
- YxListener.java
- YxParser.java
- YxVisitor.java

- Yx.tokens ANTLR会给我们定义的词法符号指定一个数字形式的类型，然后将他们的对应关系存储到该文件当中。
- YxLexer.java 词法解析类
- YxParser.java 语法解析类
- YxListener.java, YxBaseListener.java 监听器
- YxVisitor.java, YxBaseVisor.java 访问器

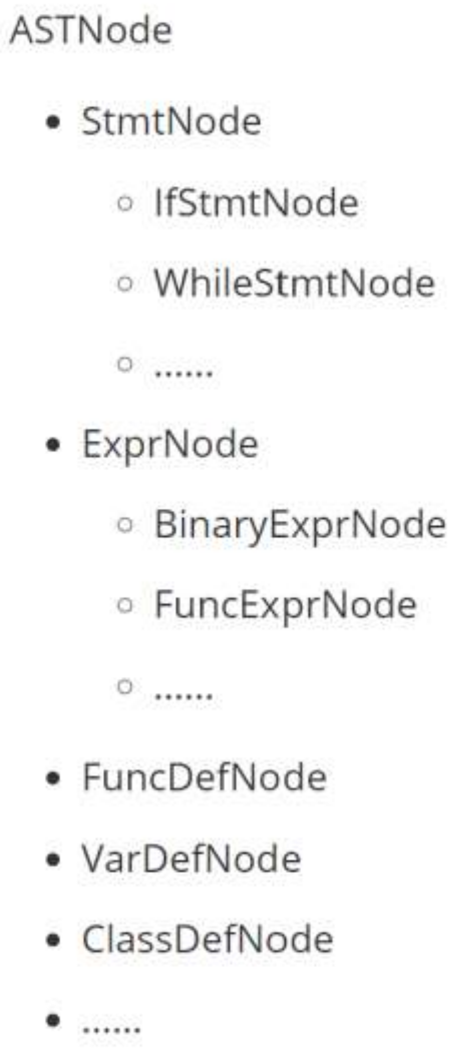
监听器和访问器是antlr4提供的两种遍历方法。两者的具体意义《antlr4权威指南》(49页起)有详细说明。

ASTBuilder

antlr4 生成的代码为我们构建了一棵 parse tree，但是它保存的信息不足以满足我们的需求，故我们仍然需要建立一个 AST，读取 parse tree 并增添我们需要的内容。

```
public class ASTBuilder extends YxBaseVisitor<ASTNode> {

    private globalScope gScope;
    public ASTBuilder(globalScope gScope) {
        this.gScope = gScope;
    }

    Type intType, boolType;

    @Override public ASTNode visitProgram(YxParser.ProgramContext ctx) {
        RootNode root = new RootNode(new position(ctx), (FnRootNode)visit(ctx.mainFn()));
        ctx.classDef().forEach(cd -> root.strDefs.add((structDefNode) visit(cd)));
        return root;
    }

    @Override public ASTNode visitClassDef(YxParser.ClassDefContext ctx) {
        structDefNode structDef = new structDefNode(new position(ctx), ctx.Identifier().toString());
        ctx.varDef().forEach(vd -> structDef.varDefs.add((varDefStmtNode) visit(vd)));
        return structDef;
    }
```

ASTNode继承派生关系

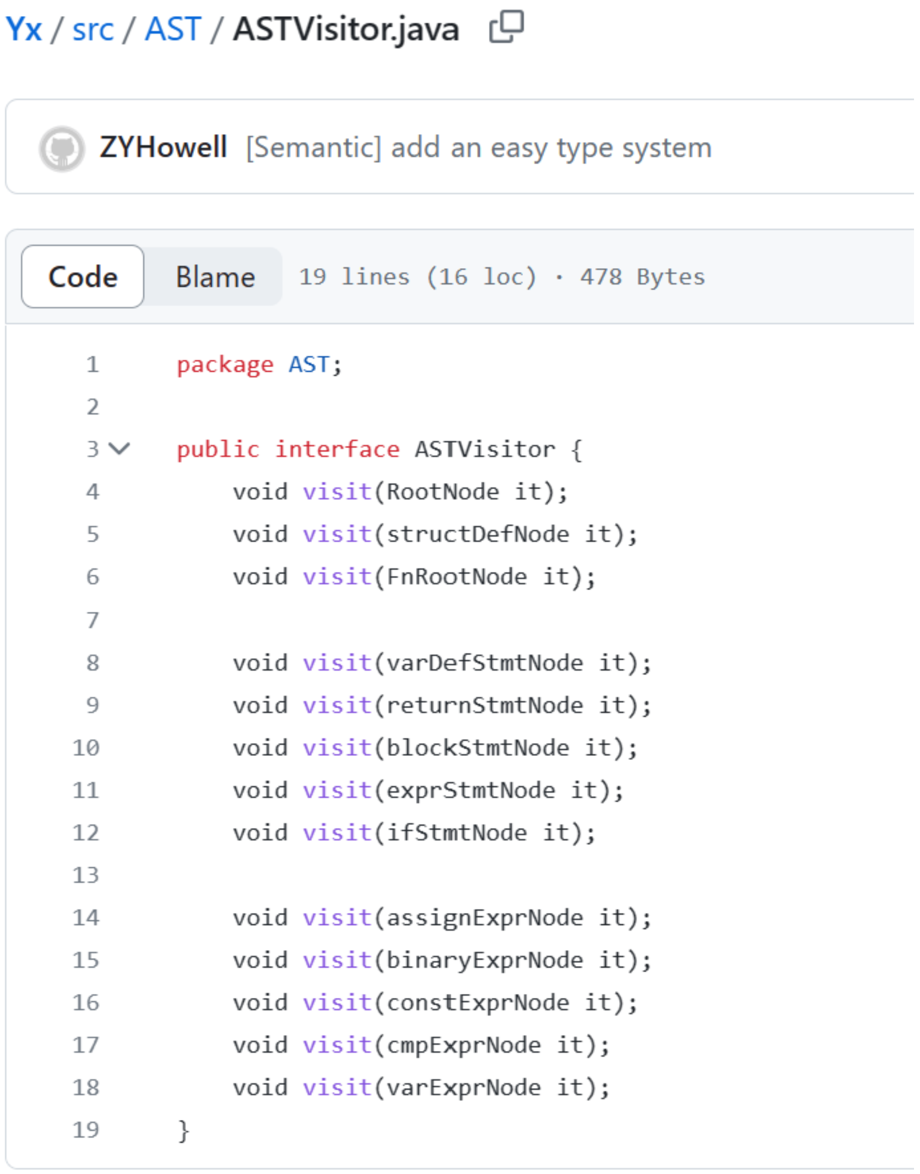


ASTNode

- StmtNode
  - IfStmtNode
  - WhileStmtNode
  - ……
- ExprNode
  - BinaryExprNode
  - FuncExprNode
  - ……
- FuncDefNode
- VarDefNode
- ClassDefNode
- ……

和g4一致

如何访问AST的节点

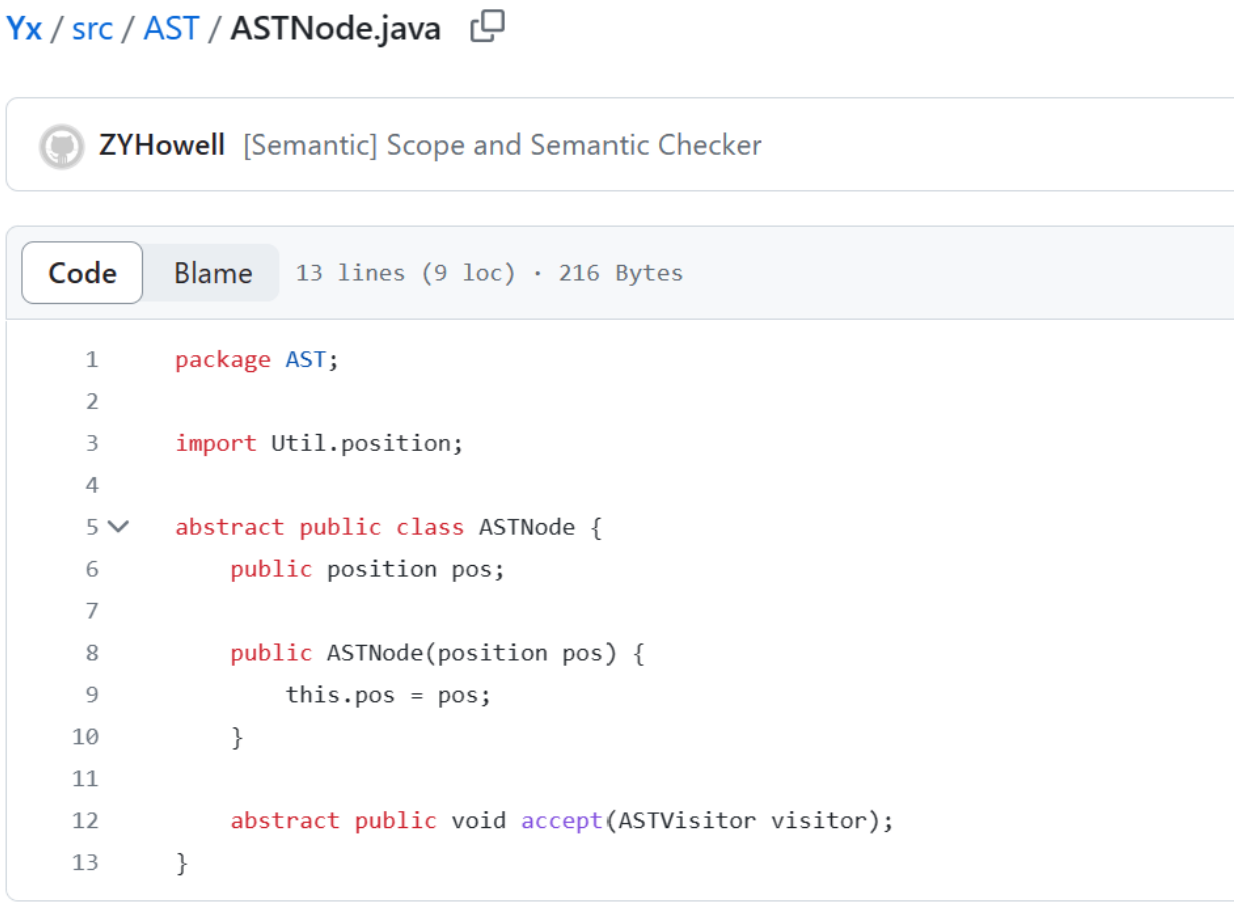


Yx / src / AST / ASTVisitor.java

ZYHowell [Semantic] add an easy type system

Code Blame 19 lines (16 loc) · 478 Bytes

```java
package AST;

public interface ASTVisitor {
    void visit(RootNode it);
    void visit(structDefNode it);
    void visit(FnRootNode it);

    void visit(varDefStmtNode it);
    void visit(returnStmtNode it);
    void visit(blockStmtNode it);
    void visit(exprStmtNode it);
    void visit(ifStmtNode it);

    void visit(assignExprNode it);
    void visit(binaryExprNode it);
    void visit(constExprNode it);
    void visit(cmpExprNode it);
    void visit(varExprNode it);
}
```



Yx / src / AST / ASTNode.java

ZYHowell [Semantic] Scope and Semantic Checker

Code Blame 13 lines (9 loc) · 216 Bytes

```java
package AST;

import Util.position;

abstract public class ASTNode {
    public position pos;

    public ASTNode(position pos) {
        this.pos = pos;
    }

    abstract public void accept(ASTVisitor visitor);
}
```

各个节点的内容

```java
public class ifStmtNode extends StmtNode {
    public ExprNode condition;
    public StmtNode thenStmt, elseStmt;

    public ifStmtNode(ExprNode condition, StmtNode thenStmt, StmtNode elseStmt, position pos) {
        super(pos);
        this.condition = condition;
        this.thenStmt = thenStmt;
        this.elseStmt = elseStmt;
    }

    @Override
    public void accept(ASTVisitor visitor) {
        visitor.visit(this);
    }
}
```

各个节点的内容

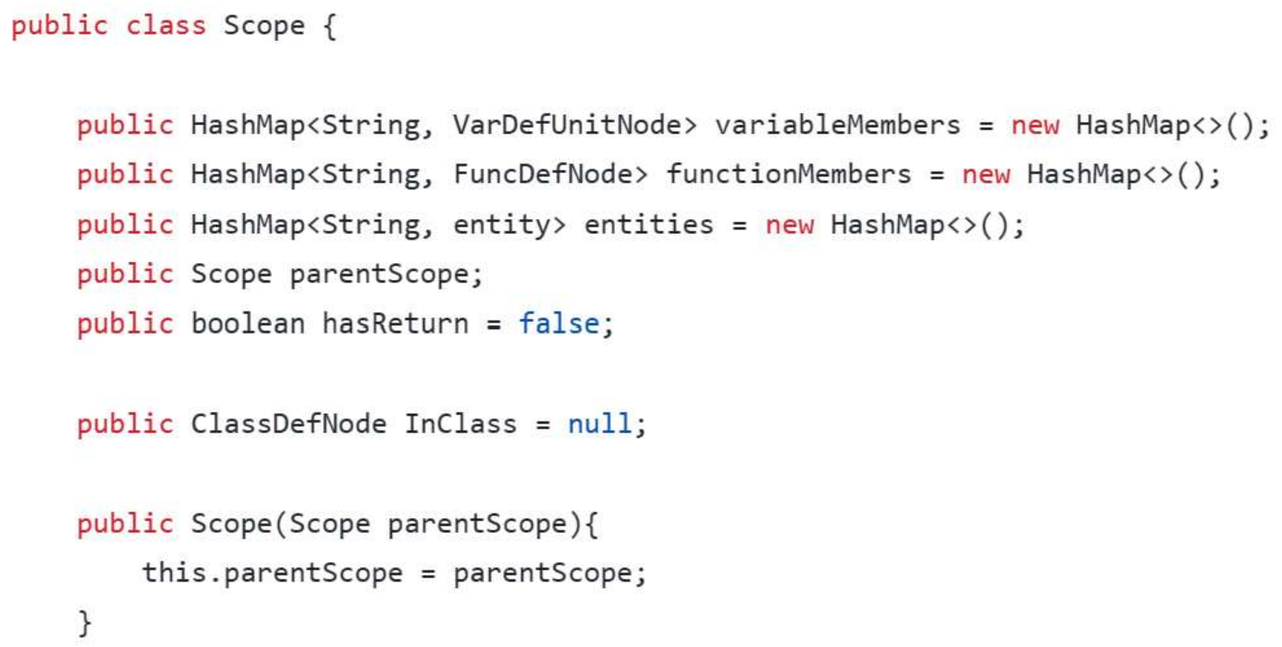
```java
public class binaryExprNode extends ExprNode {
    public ExprNode lhs, rhs;
    public enum binaryOpType {
        add, sub
    }
    public binaryOpType opCode;

    public binaryExprNode(ExprNode lhs, ExprNode rhs, binaryOpType opCode, Type intType, position pos) {
        super(pos);
        this.lhs = lhs;
        this.rhs = rhs;
        this.opCode = opCode;
        type = intType;
    }

    @Override
    public void accept(ASTVisitor visitor) {
        visitor.visit(this);
    }
}
```

Next step:

SymbolCollector
and
SemanticChecker

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)。



```java
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();
    }
}
```

SymbolCollector

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

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

SymbolCollector 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），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

Type system：
需要处理的问题：
基本类型（int、bool等）、自定义类？数组？

Yx

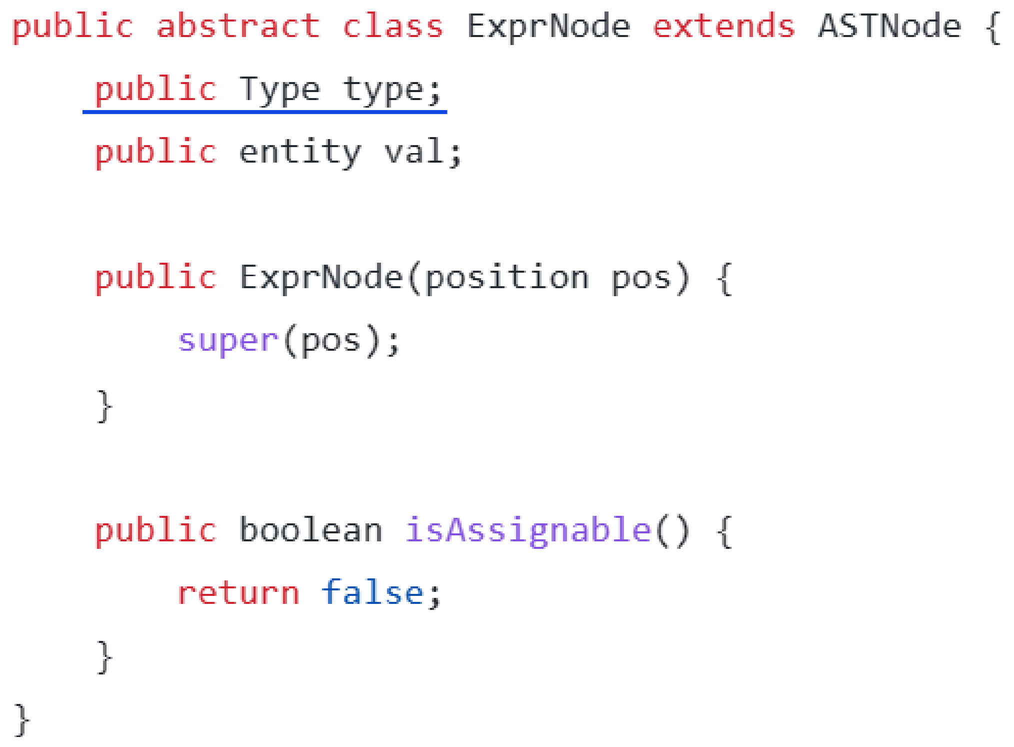
```
package Util;

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);
    }

}
```

SymbolCollector

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;
    }
}
```

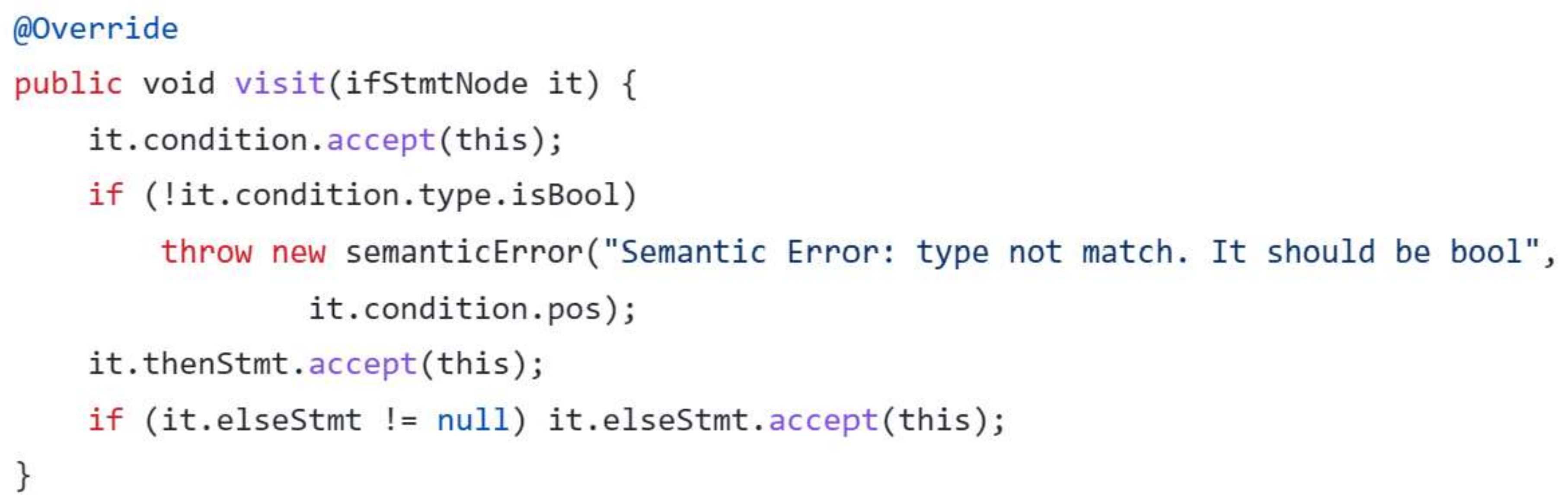
e.g.

1. a = b + c;

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

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。



```java
@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);
}
```

Q × A