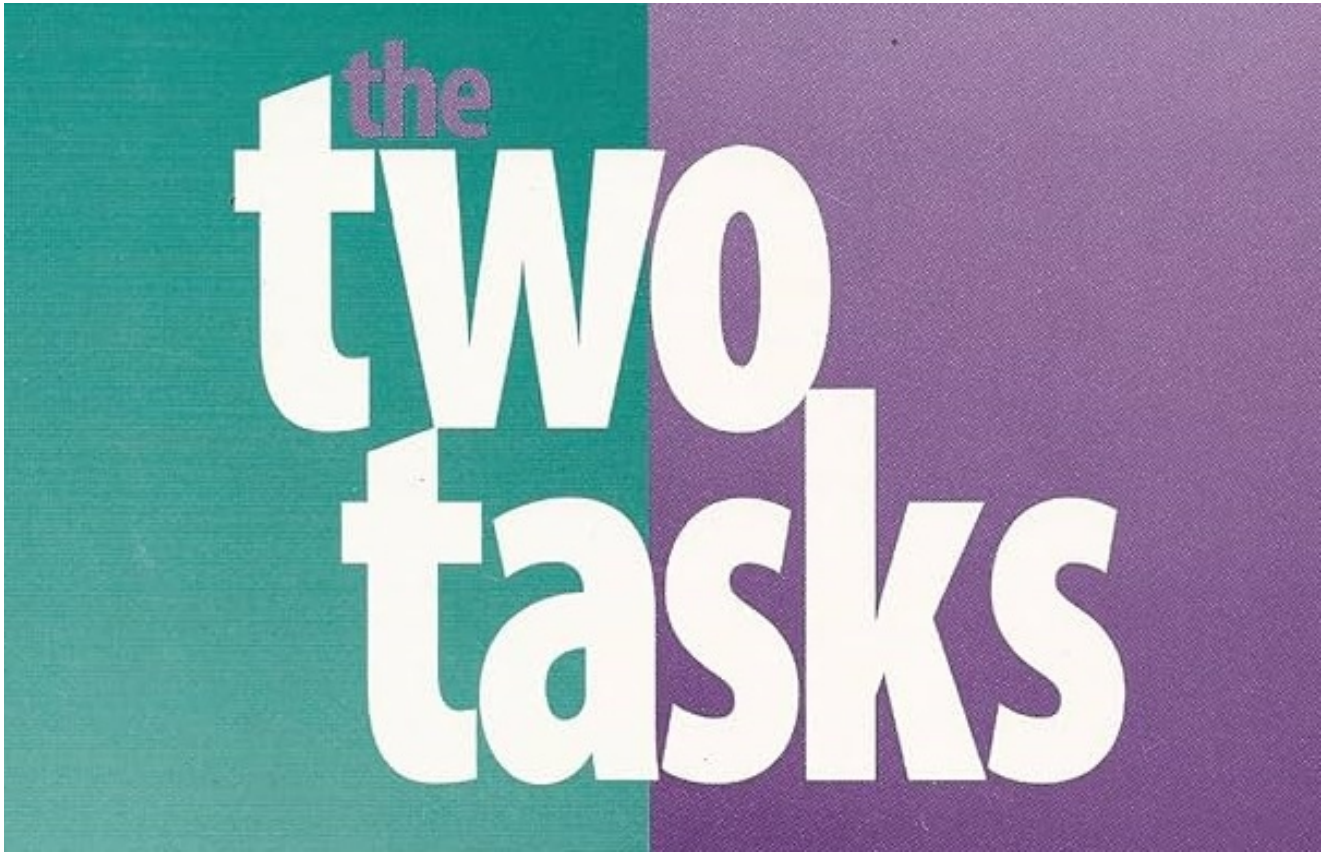
# 二、语法分析
# (4. ANTLR 4 语法分析器)

魏恒峰

hfwei@nju.edu.cn

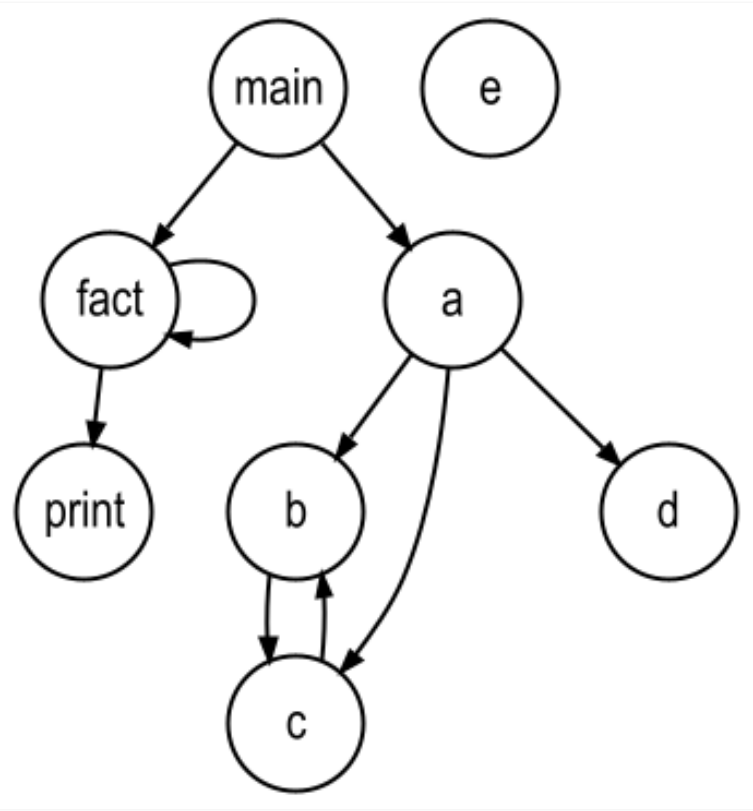
2024 年 03 月 22 日


the
two
tasks

## 任务一： 设计一个类 C 语言 Cymbol.g4

```
1   int factorial(int n) {
2     if (n == 1)
3     then return 1;
4
5     return n * factorial(n - 1);
6   }
7
8   int main() {
9     factorial(5);
10  }
```

## 任务二： 抽取函数调用图 (Function Call Graph)




**无奖竞猜:** 我们需要写多少行 Java 代码? 约 5 行核心代码

# Cymbol.g4

二义性 (ambiguous) 文法

# IfStat.g4

```
stat : 'if' expr 'then' stat
     | 'if' expr 'then' stat 'else' stat
     | expr
     ;
```

```
if a then if b then c else d
```

```
if a then if b then c else d
```

# IfStat.g4

```
stat : 'if' expr 'then' stat
     | 'if' expr 'then' stat 'else' stat
     | expr
     ;
```

# IfStatOpenMatched.g4

```
stat : matched_stat | open_stat ;

matched_stat : 'if' expr 'then' matched_stat 'else' matched_stat
             | expr
             ;

open_stat: 'if' expr 'then' stat
         | 'if' expr 'then' matched_stat 'else' open_stat
         ;
```

# Expr.g4

```
expr :
    | expr '*' expr
    | expr '-' expr
    | DIGIT
    ;
```

运算符的**结合性**带来的**二义性**

# ExprAssoc.g4

```
expr: '!' expr
    | <assoc = right> expr '^' expr
    | DIGIT
    ;
```

右结合运算符、前缀运算符与后缀运算符的结合性

# Expr.g4

```
expr :
    | expr '*' expr
    | expr '-' expr
    | DIGIT
    ;
```

运算符的**优先级**带来的**二义性**

# ExprLR.g4

```
expr : expr '-' term
     | term
     ;

term : term '*' factor
     | factor
     ;

factor : DIGIT ;
```

左递归 (左结合)

```
expr :
     | expr '*' expr
     | expr '-' expr
     | DIGIT
     ;
```

ANTLR 4 可以处理该文法

```
expr :
   | expr '*' expr
   | expr '-' expr
   | DIGIT
   ;
```

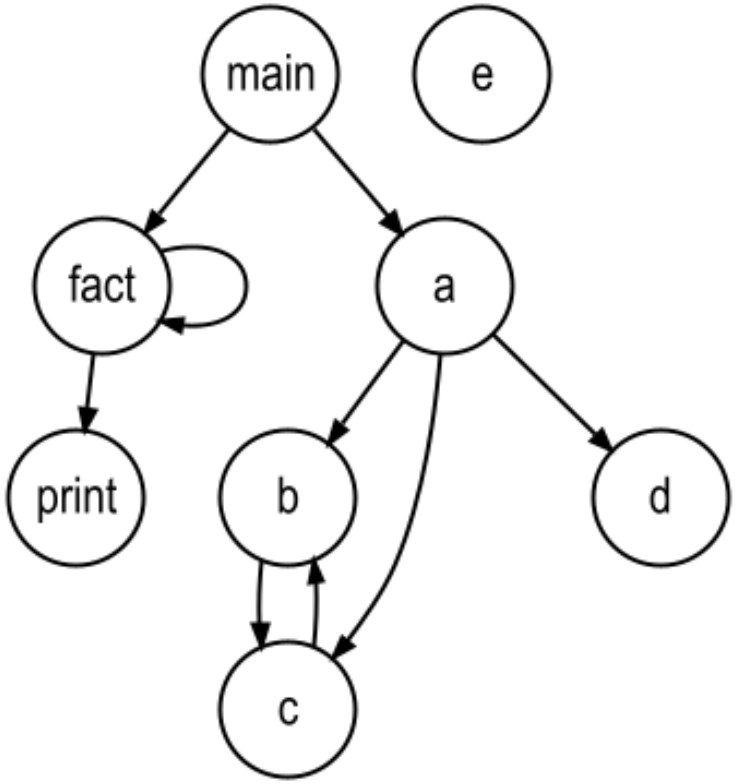
ANTLR 4 可以处理该文法

## ExprRR.g4

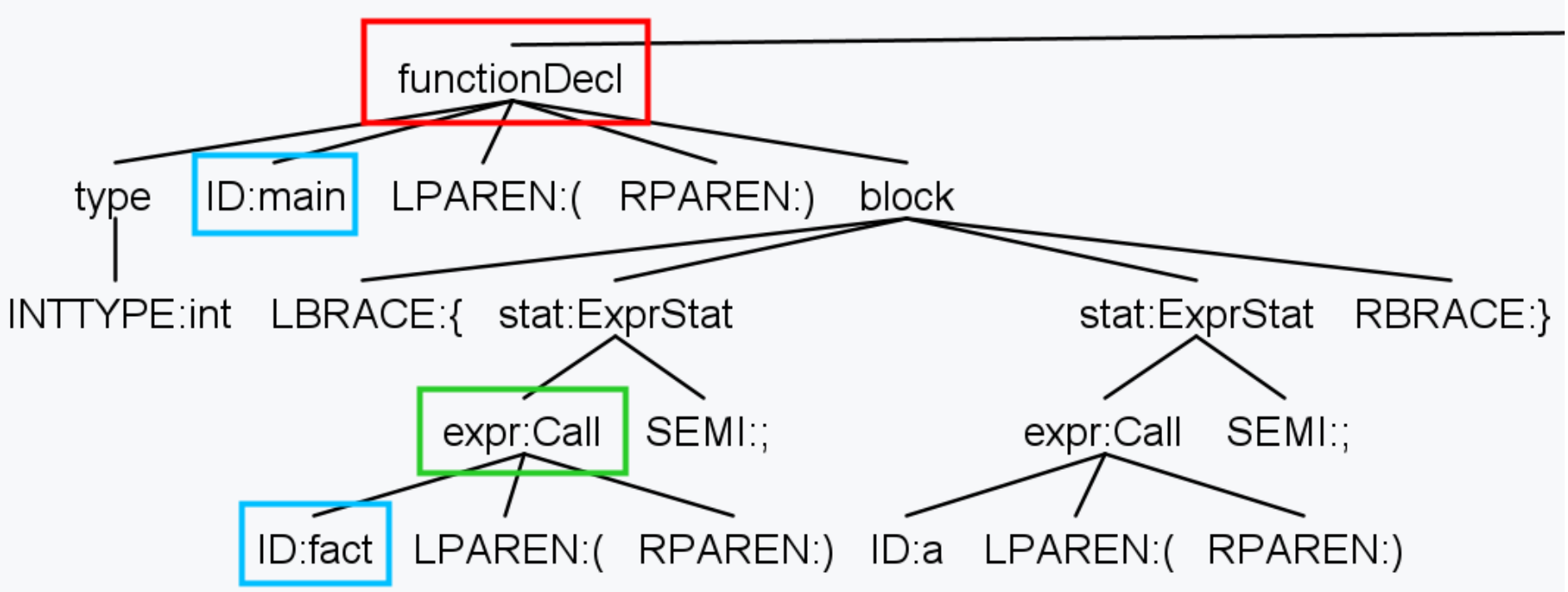
```
expr : term expr_prime ;
expr_prime : '-' term expr_prime
     |
     ;

term : factor term_prime ;
term_prime : '*' factor term_prime
     |
     ;

factor : DIGIT ;
```

右递归 (右结合)

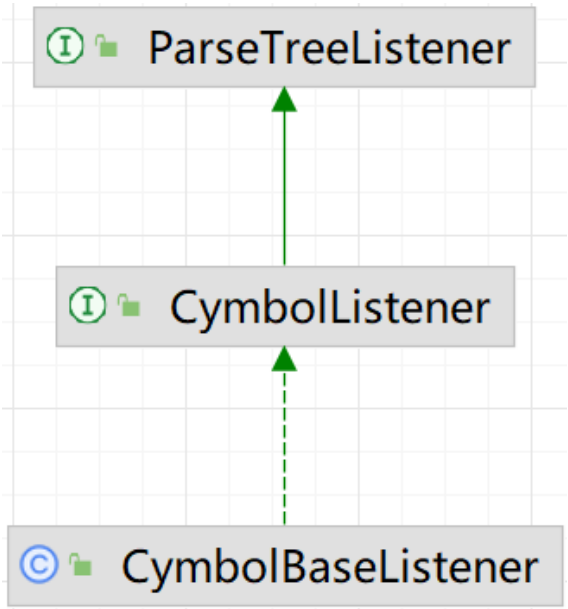
# Function Call Graphs



ParseTreeWalker 负责以 **DFS** 方式自动遍历语法树




Listener 负责监听**进入**、**退出**节点事件




| | |
|---|---|
| enterExprStat(ExprStatContext) | void |
| enterFormalParameter(FormalParameterContext) | void |
| enterFormalParameters(FormalParametersContext) | void |
| enterFunctionDecl(FunctionDeclContext) | void |
| enterId(IdContext) | void |
| enterIfStat(IfStatContext) | void |
| enterIndex(IndexContext) | void |
| enterInt(IntContext) | void |
| enterMultDiv(MultDivContext) | void |
| enterNegate(NegateContext) | void |
| enterNot(NotContext) | void |
| enterParens(ParensContext) | void |
| enterPower(PowerContext) | void |
| enterProg(ProgContext) | void |
| enterReturnStat(ReturnStatContext) | void |
| enterType(TypeContext) | void |
| enterVarDecl(VarDeclContext) | void |
| enterVarDeclStat(VarDeclStatContext) | void |
| exitAddSub(AddSubContext) | void |
| exitAssignStat(AssignStatContext) | void |
| exitBlock(BlockContext) | void |
| exitBlockStat(BlockStatContext) | void |
| exitCall(CallContext) | void |
| exitEQNE(EQNEContext) | void |
| exitExprList(ExprListContext) | void |
| exitExprStat(ExprStatContext) | void |
| exitFormalParameter(FormalParameterContext) | void |
| exitFormalParameters(FormalParametersContext) | void |
| exitFunctionDecl(FunctionDeclContext) | void |
| exitId(IdContext) | void |
| exitIfStat(IfStatContext) | void |

# Timing (时机) !!!

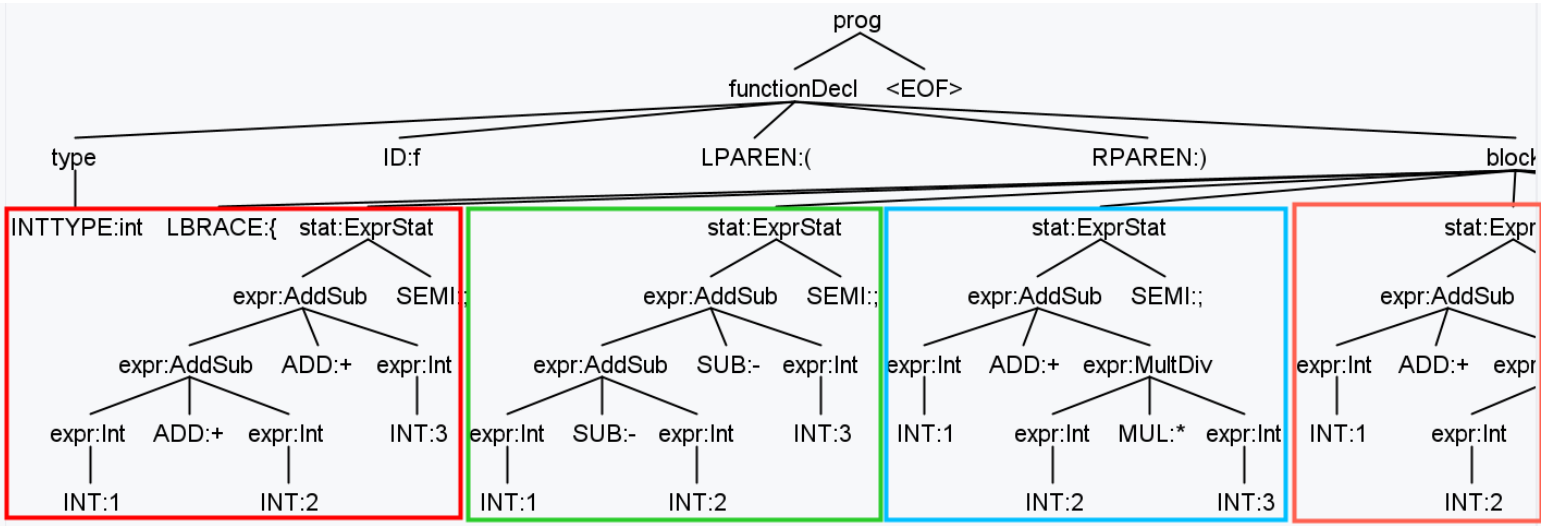
能否将 `enterFunctionDecl` 换成 `exitFunctionDecl`?

能否将 `enterFunctionCall` 换成 `exitFunctionCall`?

# ParseTreeWalker 与 Listener

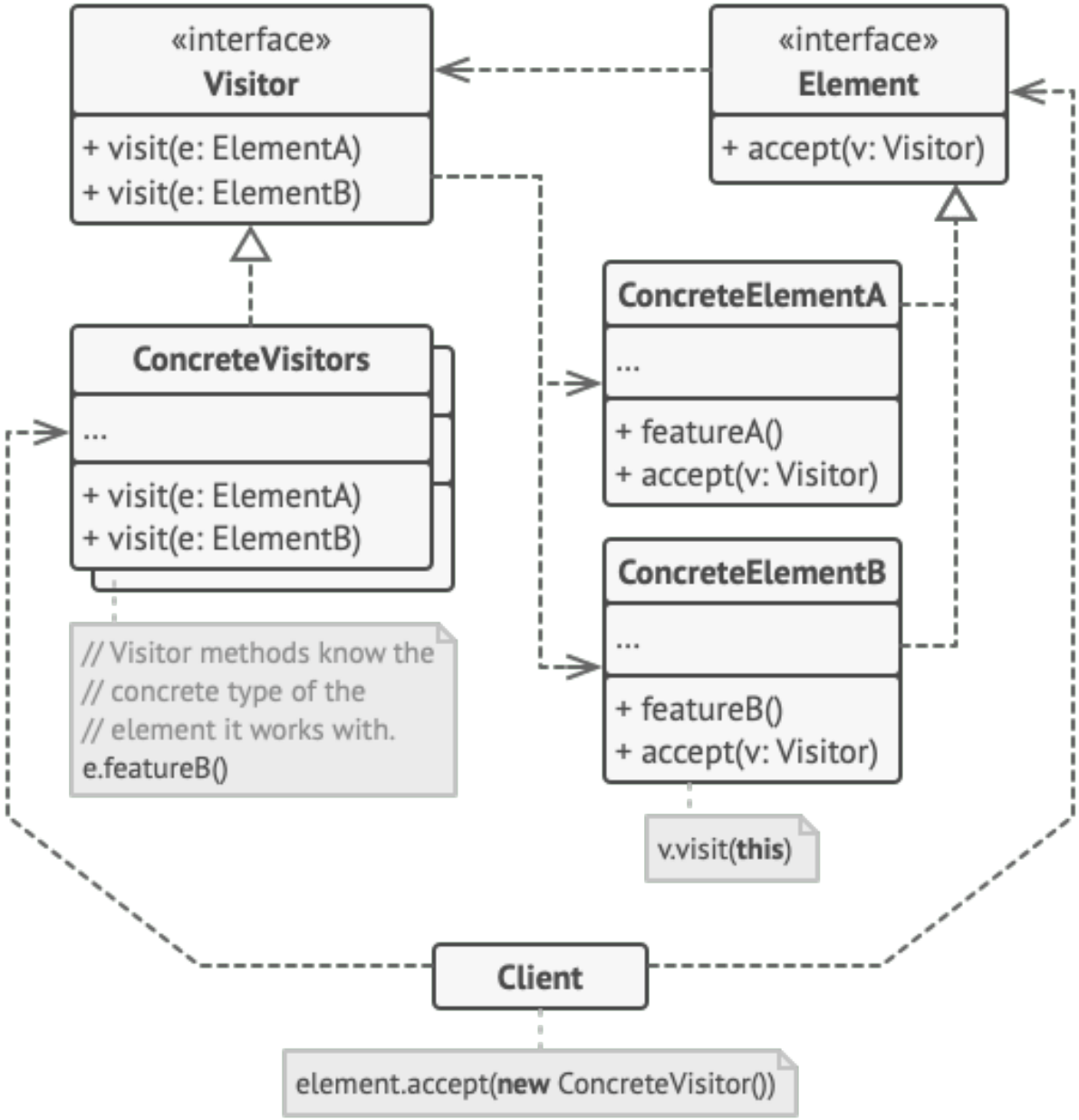
```
23  public void walk(ParseTreeListener listener, ParseTree t) {
24    if ( t instanceof ErrorNode) {
25      listener.visitErrorNode((ErrorNode)t);
26      return;
27    }
28    else if ( t instanceof TerminalNode) {
29      listener.visitTerminal((TerminalNode)t);
30      return;
31    }
32    RuleNode r = (RuleNode)t;
33        enterRule(listener, r);
34        int n = r.getChildCount();
35        for (int i = 0; i<n; i++) {
36            walk(listener, r.getChild(i));
37        }
38    exitRule(listener, r);
39    }
```

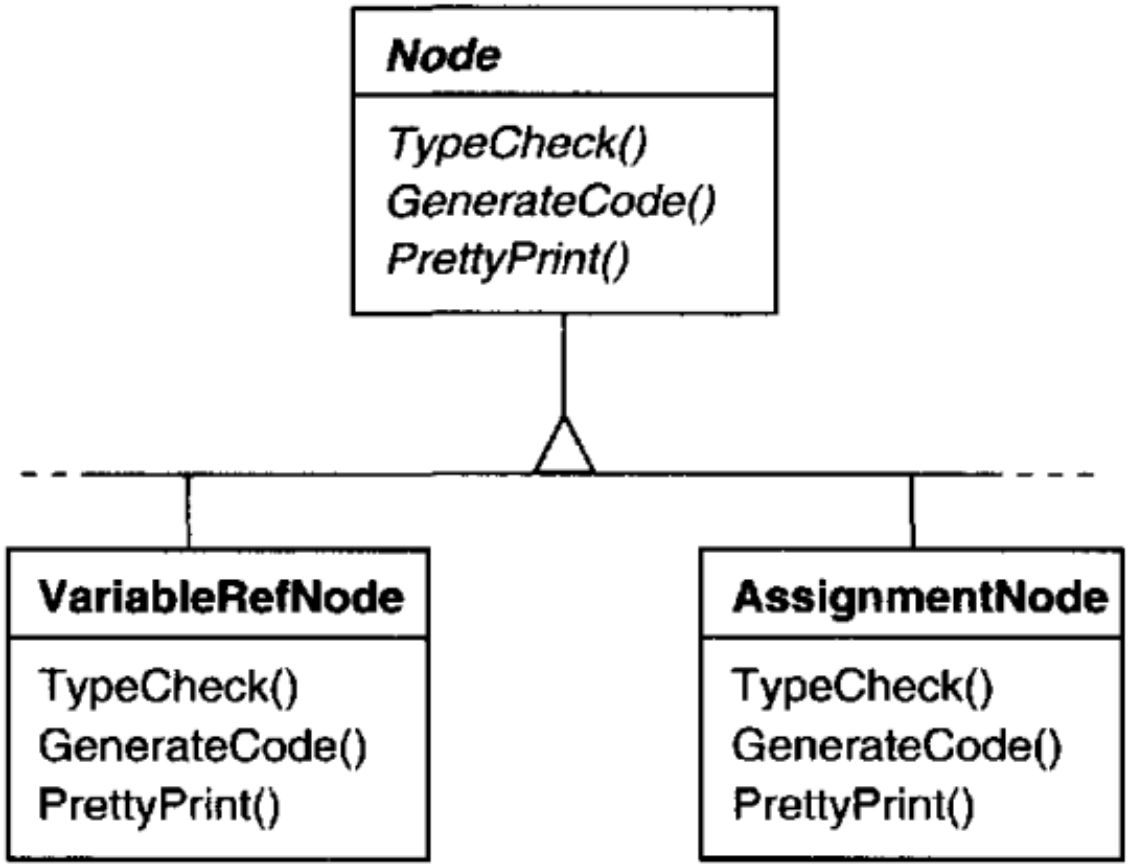
```
1  int f() {
2    1 + 2 + 3;
3    1 - 2 - 3;
4    1 + 2 * 3;
5    1 + 2 / 3;
6    ----1;
7    (1 + 2) * 3;
8    1 ^ 2 ^ 3;
9  }
```

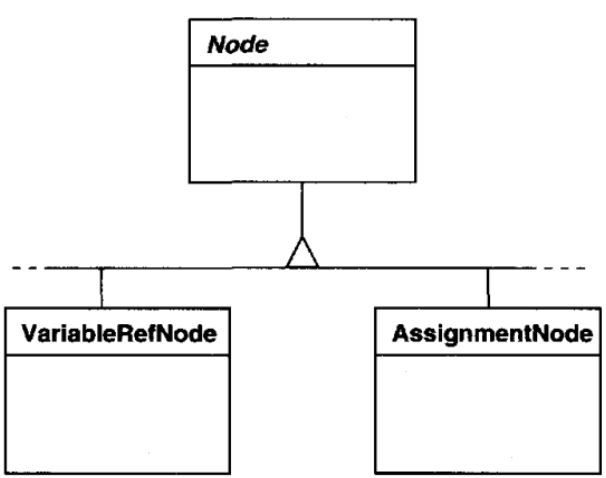


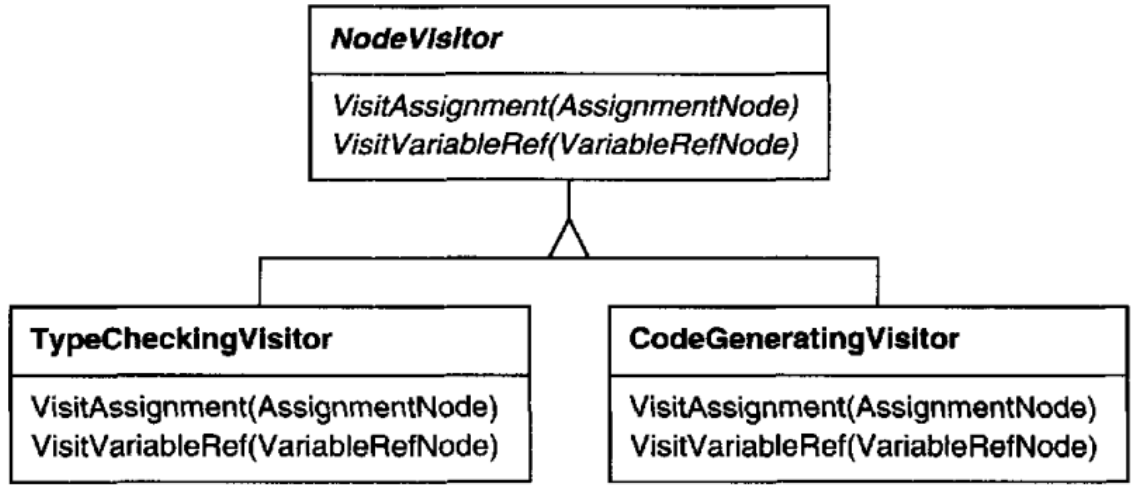

Annotated Parse Tree (标注语法分析树)

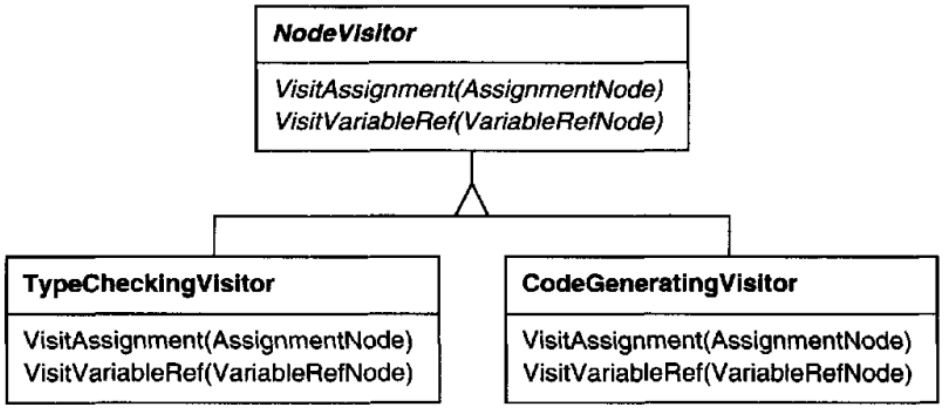
《ANTLR 4 权威指南》第 7.5.3 节

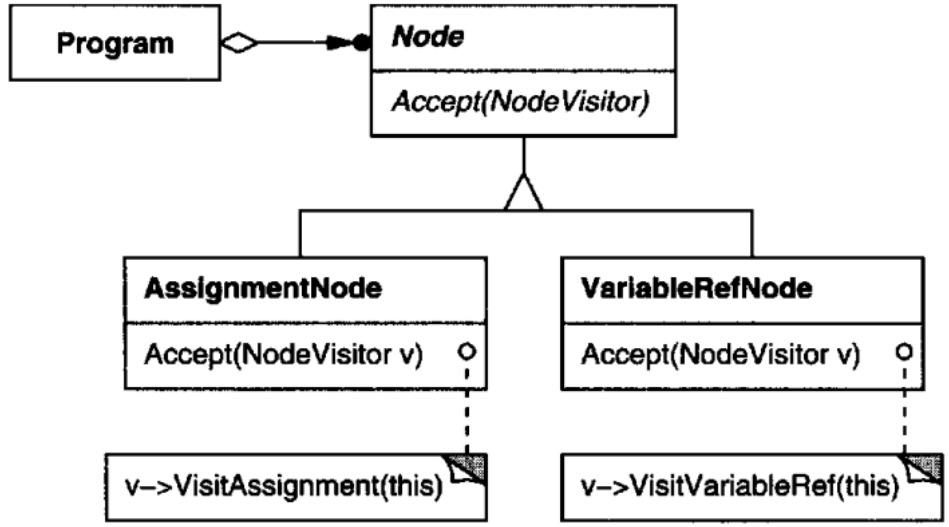

«interface»
Visitor
+ visit(e: ElementA)
+ visit(e: ElementB)
«interface»
Element
+ accept(v: Visitor)
ConcreteVisitors
...
+ visit(e: ElementA)
+ visit(e: ElementB)
// Visitor methods know the
// concrete type of the
// element it works with.
e.featureB()
ConcreteElementA
...
+ featureA()
+ accept(v: Visitor)
ConcreteElementB
...
+ featureB()
+ accept(v: Visitor)
v.visit(this)
Client
element.accept(new ConcreteVisitor())

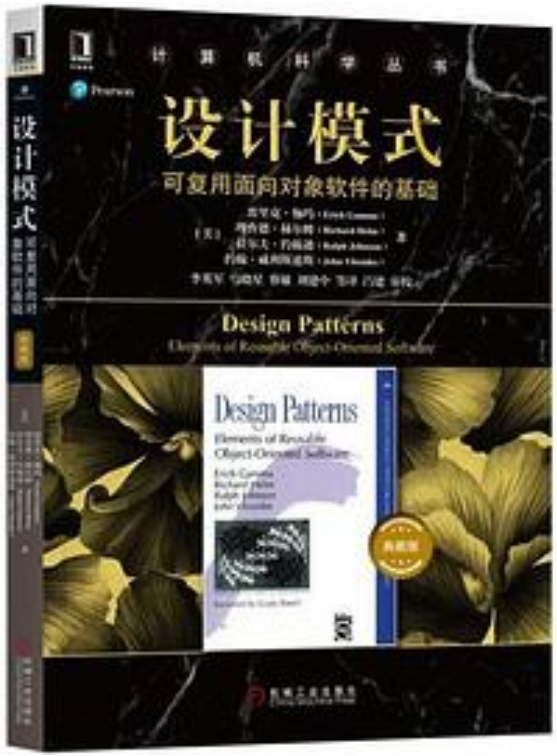

Node
TypeCheck()
GenerateCode()
PrettyPrint()
VariableRefNode
TypeCheck()
GenerateCode()
PrettyPrint()
AssignmentNode
TypeCheck()
GenerateCode()
PrettyPrint()

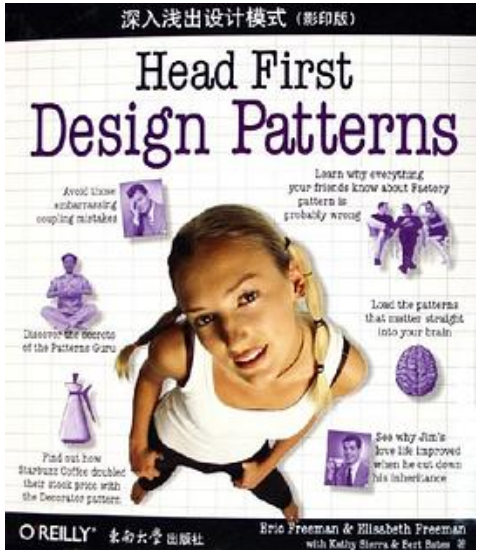

Node
VariableRefNode
AssignmentNode


NodeVisitor
VisitAssignment(AssignmentNode)
VisitVariableRef(VariableRefNode)
TypeCheckingVisitor
VisitAssignment(AssignmentNode)
VisitVariableRef(VariableRefNode)
CodeGeneratingVisitor
VisitAssignment(AssignmentNode)
VisitVariableRef(VariableRefNode)


NodeVisitor
VisitAssignment(AssignmentNode)
VisitVariableRef(VariableRefNode)
TypeCheckingVisitor
VisitAssignment(AssignmentNode)
VisitVariableRef(VariableRefNode)
CodeGeneratingVisitor
VisitAssignment(AssignmentNode)
VisitVariableRef(VariableRefNode)

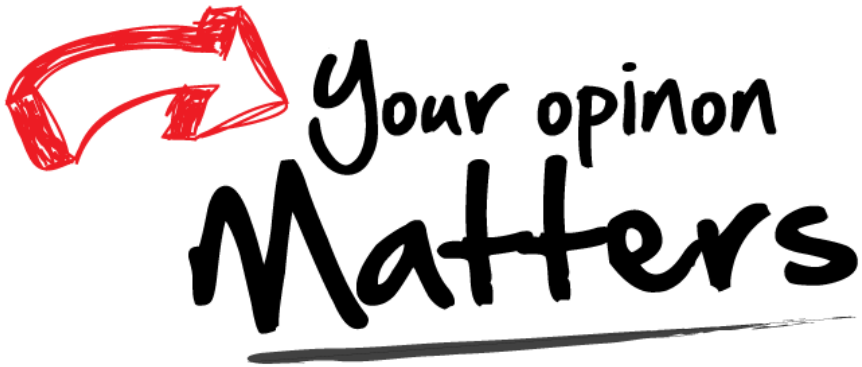

Program
Node
Accept(NodeVisitor)
AssignmentNode
Accept(NodeVisitor v)
VariableRefNode
Accept(NodeVisitor v)
v->VisitAssignment(this)
v->VisitVariableRef(this)


计 算 机 科 学 丛 书
设计模式
可复用面向对象软件的基础
Design Patterns


深入浅出设计模式（影印版）
Head First
Design Patterns
O'REILLY
Eric Freeman & Elisabeth Freeman


Java示例代码
图解设计模式
原版连续畅销12年、重印25次!
194张图表轻松理解GoF的23种设计模式


Thank
You!




Office 926

hfwei@nju.edu.cn