

Git Graph



解析器Parser

```
// 程序：主方法 类声明列表
// PROG: MAINMETHOD CLASSDECLLIST

// 主方法：public int main() { 变量声明列表 语句列表 }
// MAINMETHOD: PUBLIC INT MAIN '(' ')' '{' VARDECLLIST STMLIST '}';

// 类声明：类名 [基类名] { 变量声明列表 方法声明列表 }
// CLASSDECL: PUBLIC CLASS ID '{' VARDECLLIST METHODDECLLIST '}'
//           | PUBLIC CLASS ID EXTENDS ID '{' VARDECLLIST METHODDECLLIST '}'

// 类型： 整型 | 整型数组 | 类
// TYPE: INT | INT '[' ']' | CLASS ID

// 变量声明
// VARDECL: CLASS ID ID ';'
//         | INT ID ';'
//         | INT '[' ']' ID ';'
//         | INT '[' NUM ']' ID ';'
//         | INT ID '=' CONST ';'

```

```
//          | INT '[' ']' ID '=' '{' CONSTLIST '}' ';'
//          | INT '[' NUM ']' ID '=' '{' CONSTLIST '}' ';'

// 方法声明：返回类型 方法名(形参列表) { 变量声明列表 语句列表 }
// METHODDECL: PUBLIC TYPE ID '(' FORMALLIST ')' '{' VARDECLLIST STMLIST '}'

// 语句
// STM: '{' STMLIST '}'
//          | IF '(' EXP ')' STM ELSE STM
//          | IF '(' EXP ')' STM
//          | WHILE '(' EXP ')' STM
//          | WHILE '(' EXP ')' ';'
//          | EXP '=' EXP ';'
//          | EXP '.' ID '(' EXPLIST ')' ';'
//          | CONTINUE ';'
//          | BREAK ';'
//          | RETURN EXP ';'
//          | PUTINT '(' EXP ')' ';'
//          | PUTCH '(' EXP ')' ';'
//          | PUTARRAY '(' EXP ',' EXP ')' ';'
//          | STARTTIME '(' ')' ';'
//          | STOPTIME '(' ')' ';'

// 表达式
// EXP -> '(' EXP ')'
//          | '(' '{' STMLIST '}' EXP ')'
//          | ID
//          | NUM
//          | TRUE | FALSE
//          | EXP '[' EXP ']'
//          | OP
//          | EXP [+ - * / COMP && ||] EXP
//          | [-!] EXP
//          | THIS
//          | EXP '.' ID '(' EXPLIST ')'
//          | EXP '.' ID
//          | GETINT '(' ')'
//          | GETCH '(' ')'
//          | GETARRAY '(' EXP ')'
//          | LENGTH '(' EXP ')'
```

类继承关系处理

```
void AST_Name_Map_Visitor::visit(ClassDecl* node)
```

父类在子类前已声明

子类从其所有祖先类继承类变量

子类不允许覆盖父类变量

-> 因此在初始化类时, 先映射父类的成员变量

子类从其所有祖先类继承类方法

子类可以覆盖其祖先类中声明的方法, 但必须具有相同(返回类型, 参数类型)

-> 因此等到子类的方法初始完之后, 再映射父类方法, 并检查是否可覆盖

同名变量优先级处理

```
void AST_Semant_Visitor::visit(IdExp* node)
```

1. 对于类成员访问 `ClassVar`, 将 `is_fetch_class_var=true`
并设置 `fetch_class_name = get<string>(obj_semant->get_type_par())`
使得语义处理id时, 指向为该类的成员变量
2. 对于类方法调用 `CallStm` 和 `CallExp`, 将 `is_fetch_class_method=false`
并设置 `fetch_class_name = get<string>(obj_semant->get_type_par())`
使得语义处理id时, 指向为该类的成员方法
3. 其他情况按照优先级依次判断
 - 类方法局部变量: `is_method_var()`
 - 类方法参数: `is_method_formal()`
 - 类成员变量: `is_class_var()`
 - 类方法: `is_method()`
 - 类: `is_class()`