Лабораторная работа № 2.1. Синтаксические деревья

19 февраля 2024 г.

Сергей Виленский, ИУ9-62Б

Цель работы

Целью данной работы является изучение представления синтаксических деревьев в памяти компилятора и приобретение навыков преобразования синтаксических деревьев.

Индивидуальный вариант

Любая неанонимная функция должна в начале выполнения и перед возвратом (оператором `return` или при достижении конца блока) выводить слово `"Starts "` и `"Ends "`, соответственно, своё имя и время, прошедшее с начала выполнения программы (см. функции `time.Now()` и `time.Since()`, можно использовать `defer`).

Реализация

Демонстрационная программа:

```
package main

func fib(a int) int {
    if a <= 1 {
        return 1
    }
    return fib(a-1) + fib(a-2)
}

func main() {
    fib(3)
}
```

Программа, осуществляющая преобразование синтаксического дерева:

```go
package main

import (
    "fmt"
    "go/ast"
    "go/format"
    "go/parser"
    "go/token"
    "os"
)

func insertImport(file *ast.File, importName string) {
    ast.Inspect(file, func(node ast.Node) bool {
        if File, ok := node.(*ast.File); ok {
            File.Decls = append(
                append([]ast.Decl{},
                    &ast.GenDecl{
                        Doc: nil,
                        Tok: token.IMPORT,
                        Specs: []ast.Spec{
                            &ast.ImportSpec{
                                Doc:  nil,
                                Name: nil,
                                Path: &ast.BasicLit{
                                    Kind:  token.STRING,
                                    Value: "\"" + importName + "\"",
                                },
                                Comment: nil,
                            },
                        },
                    }),
                File.Decls...,
            )
        }
        return true
    })
}

// поискать либу для генерации UUID (уникального id)
func insertTimerSet(file *ast.File) {
    ast.Inspect(file, func(node ast.Node) bool {
        if File, ok := node.(*ast.File); ok {
            File.Decls = append(
                append([]ast.Decl{},
```

```
                    &ast.GenDecl{
                        Doc: nil,
                        Tok: token.VAR,
                        Specs: []ast.Spec{
                            &ast.ValueSpec{
                                Doc: nil,
                                Names: []*ast.Ident{
                                    {
                                        Name: "____NOW____",
                                        Obj: &ast.Object{
                                            Kind: ast.ObjKind(token.VAR),
                                            Name: "____NOW____",
                                            Data: 0,
                                            Type: nil,
                                        },
                                    },
                                },
                                Type: nil,
                                Values: []ast.Expr{
                                    &ast.CallExpr{
                                        Fun: &ast.SelectorExpr{
                                            X: &ast.Ident{
                                                Name: "time",
                                                Obj:  nil,
                                            },
                                            Sel: &ast.Ident{
                                                Name: "Now",
                                                Obj:  nil,
                                            },
                                        },
                                        Args: nil,
                                    },
                                },
                            },
                        },
                    }),
                File.Decls...,
            )
        }
        return true
    })
}

func insertFuncTimer(file *ast.File) {
    ast.Inspect(file, func(node ast.Node) bool {
        if FuncDecl, ok := node.(*ast.FuncDecl); ok {
```

```
generateCallExprNode := func() *ast.CallExpr {
    return &ast.CallExpr{
        Fun: &ast.SelectorExpr{
            X: &ast.Ident{
                Name: "fmt",
                Obj:  nil,
            },
            Sel: &ast.Ident{
                Name: "Printf",
                Obj:  nil,
            },
        },
        Args: []ast.Expr{
            &ast.BasicLit{
                Kind:  token.STRING,
                Value: "\"Starts %s %s\\n\"",
            },
            &ast.BasicLit{
                Kind:  token.STRING,
                Value: "\"" + FuncDecl.Name.Name + "\"",
            },
            &ast.CallExpr{
                Fun: &ast.SelectorExpr{
                    X: &ast.CallExpr{
                        Fun: &ast.SelectorExpr{
                            X: &ast.Ident{
                                Name: "time",
                                Obj:  nil,
                            },
                            Sel: &ast.Ident{
                                Name: "Since",
                                Obj:  nil,
                            },
                        },
                        Args: []ast.Expr{
                            &ast.Ident{
                                Name: "____NOW____",
                                Obj:  nil,
                            },
                        },
                    },
                    Sel: &ast.Ident{
                        Name: "String",
                        Obj:  nil,
                    },
                },
```

```
                            Args: nil,
                        },
                    },
                }
            }
            FuncDecl.Body.List = append(
                append(append([]ast.Stmt{},
                    &ast.ExprStmt{
                        X: generateCallExprNode(),
                    },
                ),
                    &ast.DeferStmt{
                        Call: generateCallExprNode(),
                    },
                ),
                FuncDecl.Body.List...)
        }
        return true
    })
}

func main() {
    if len(os.Args) != 2 {
        fmt.Printf("usage: astprint <filename.go>\n")
        return
    }

    // Создаём хранилище данных об исходных файлах
    fset := token.NewFileSet()

    // Вызываем парсер
    if file, err := parser.ParseFile(
        fset,                 // данные об исходниках
        os.Args[1],           // имя файла с исходником программы
        nil,                  // пусть парсер сам загрузит исходник
        parser.ParseComments, // приказываем сохранять комментарии
    ); err == nil {
        insertTimerSet(file)
        insertImport(file, "fmt")
        insertImport(file, "time")
        insertFuncTimer(file)

        if format.Node(os.Stdout, fset, file) != nil {
            fmt.Printf("Formatter error: %v\n", err)
        }
```

```
        // Если парсер отработал без ошибок, печатаем дерево
        ast.Fprint(os.Stdout, fset, file, nil)
    } else {
        // в противном случае, выводим сообщение об ошибке
        fmt.Printf("Error: %v", err)
    }
}
```

Тестирование

Результат трансформации демонстрационной программы:

```
package main

import "time"
import "fmt"

var ____NOW____ = time.Now()

func fib(a int) int {
        fmt.Printf("Starts %s %s\n", "fib", time.Since(____NOW____).String())
        defer fmt.Printf("Ends %s %s\n", "fib", time.Since(____NOW____).String())
        if a <= 1 {
                return 1
        }
        return fib(a-1) + fib(a-2)
}

func main() {
        fmt.Printf("Starts %s %s\n", "main", time.Since(____NOW____).String())
        defer fmt.Printf("Ends %s %s\n", "main", time.Since(____NOW____).String())
        fib(3)
}
```

Вывод тестового примера на `stdout` (если необходимо)

```
Starts main 0s
Starts fib 509.9µs
Starts fib 509.9µs
Starts fib 509.9µs
Ends fib 1.0238ms
Starts fib 1.0238ms
Ends fib 1.0238ms
Ends fib 509.9µs
Starts fib 1.0238ms
Ends fib 1.0238ms
```

```
Ends fib 509.9µs
Ends main 504.6µs
```

Вывод

В ходе выполнения данной лабораторной работы были изучены структуры хранения данных лексических деревьев разбора текста программы на языке программирования Golang, а также методы их модификации. В результате выполнения лабораторной работы была разработана программа, выполняющая декорирование всех неанонимных функций в пользу вывода в стандартный поток отладочных данных о времени начала и завершения работы функции.