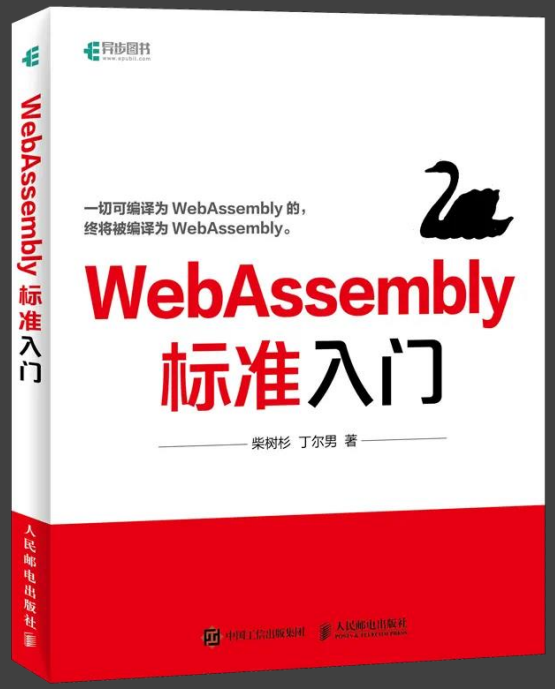
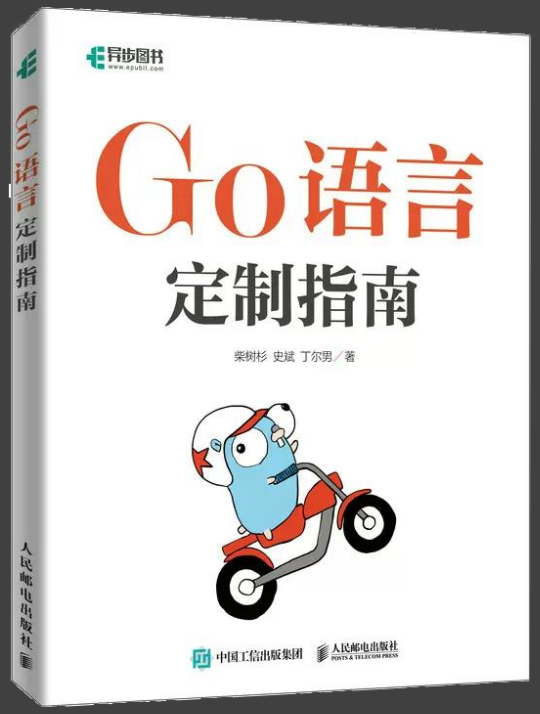
通过SSA的解释执行窥探Golang编译之一角

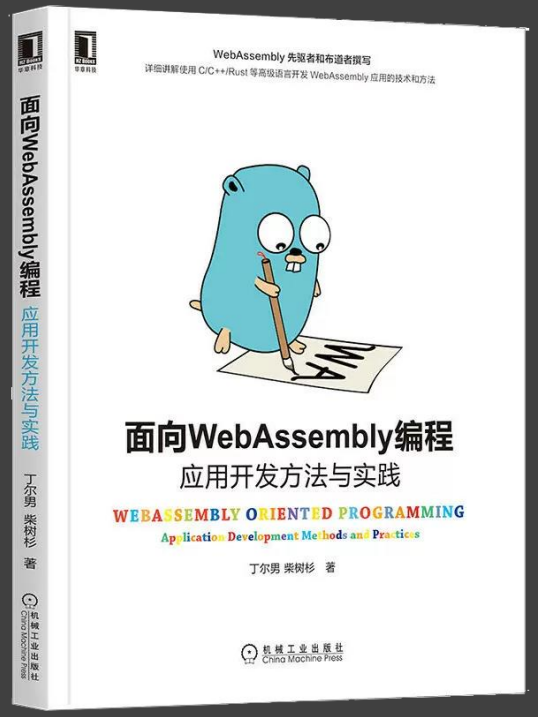
×

丁尔男

武汉航天远景 产品总监
凹语言 联合发起人
PLOC 联合发起人


异步图书
WebAssembly 标准入门
一切可编译为 WebAssembly 的，
终将被编译为 WebAssembly。
WebAssembly
标准入门
柴树杉 丁尔男 著
人民邮电出版社
中国工信出版集团
人民邮电出版社


异步图书
Go语言定制指南
人民邮电出版社
Go语言
定制指南
柴树杉 史斌 丁尔男／著
中国工信出版集团
人民邮电出版社


WebAssembly 先驱者和布道者撰写
详细讲解使用 C/C++/Rust 等高级语言开发 WebAssembly 应用的技术和方法
面向WebAssembly编程
应用开发方法与实践
丁尔男 柴树杉 著
面向WebAssembly编程
应用开发方法与实践
WEBASSEMBLY ORIENTED PROGRAMMING
Application Development Methods and Practices
丁尔男 柴树杉 著
机械工业出版社
China Machine Press

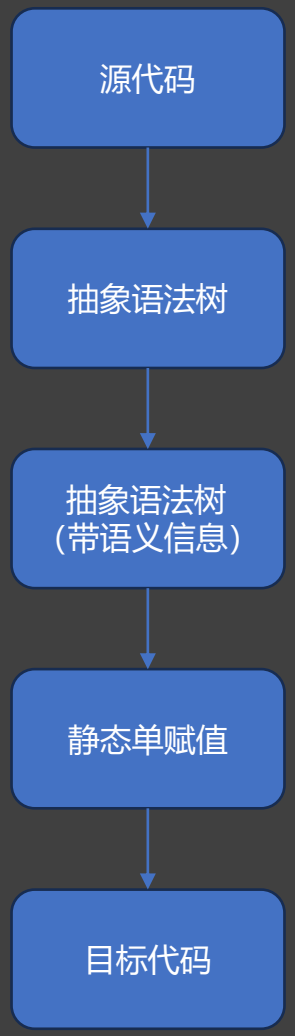
目 录

Golang 编译流程简介 01

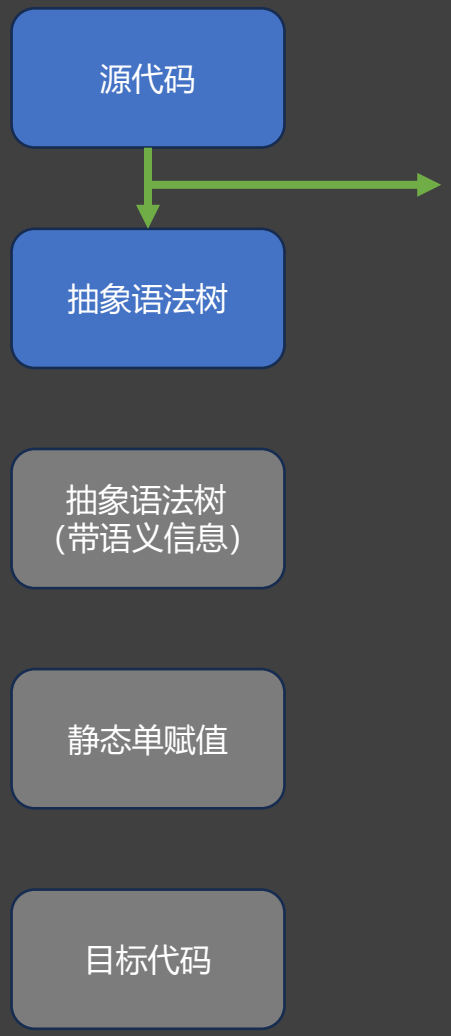
SSA 解释执行 02

基于 SSA 的应用 03

Golang 编译流程简介



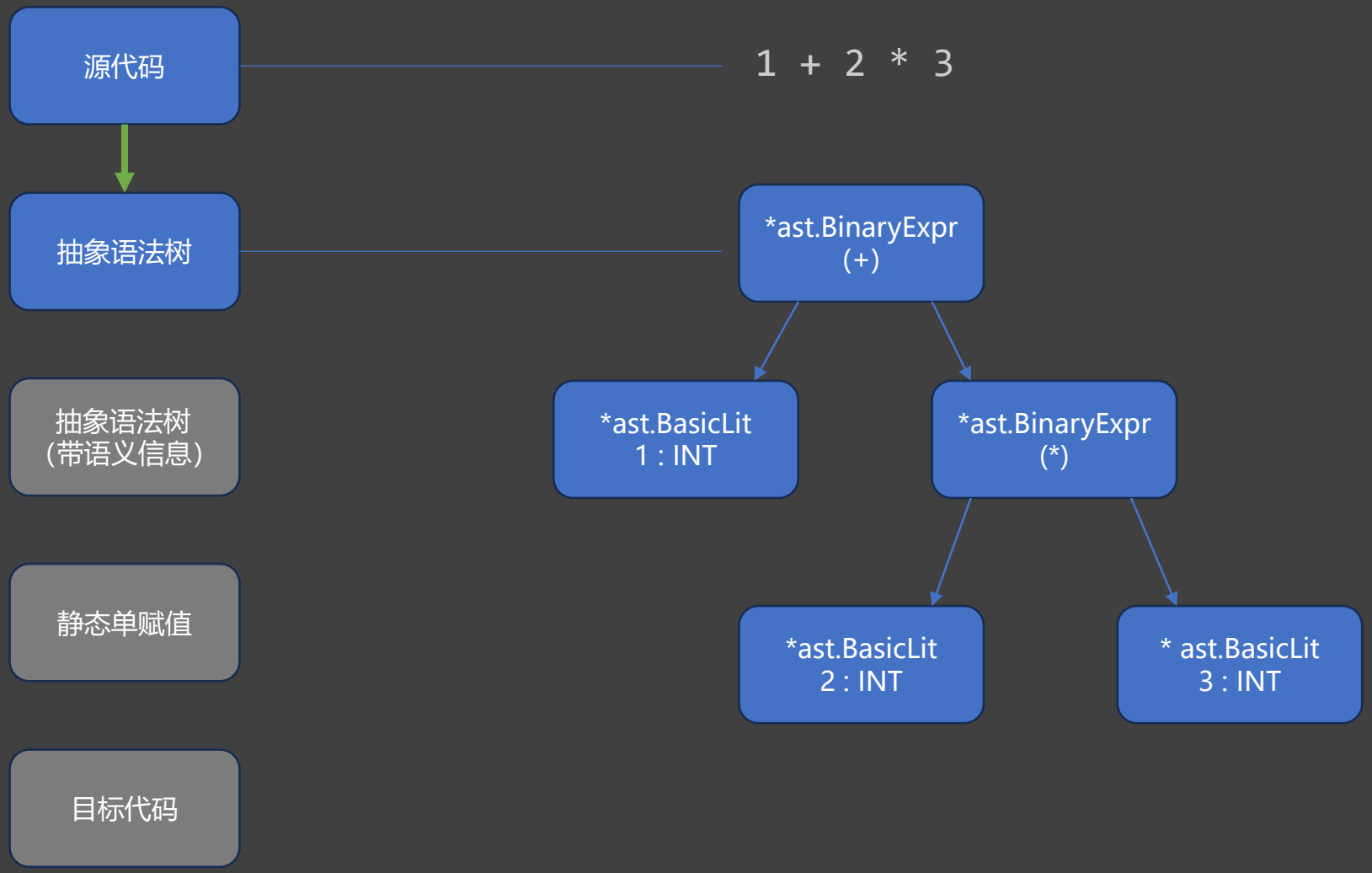
Golang 编译流程简介



语法解析，生成抽象语法树（Abstract Syntax Tree，AST），涉及的包：

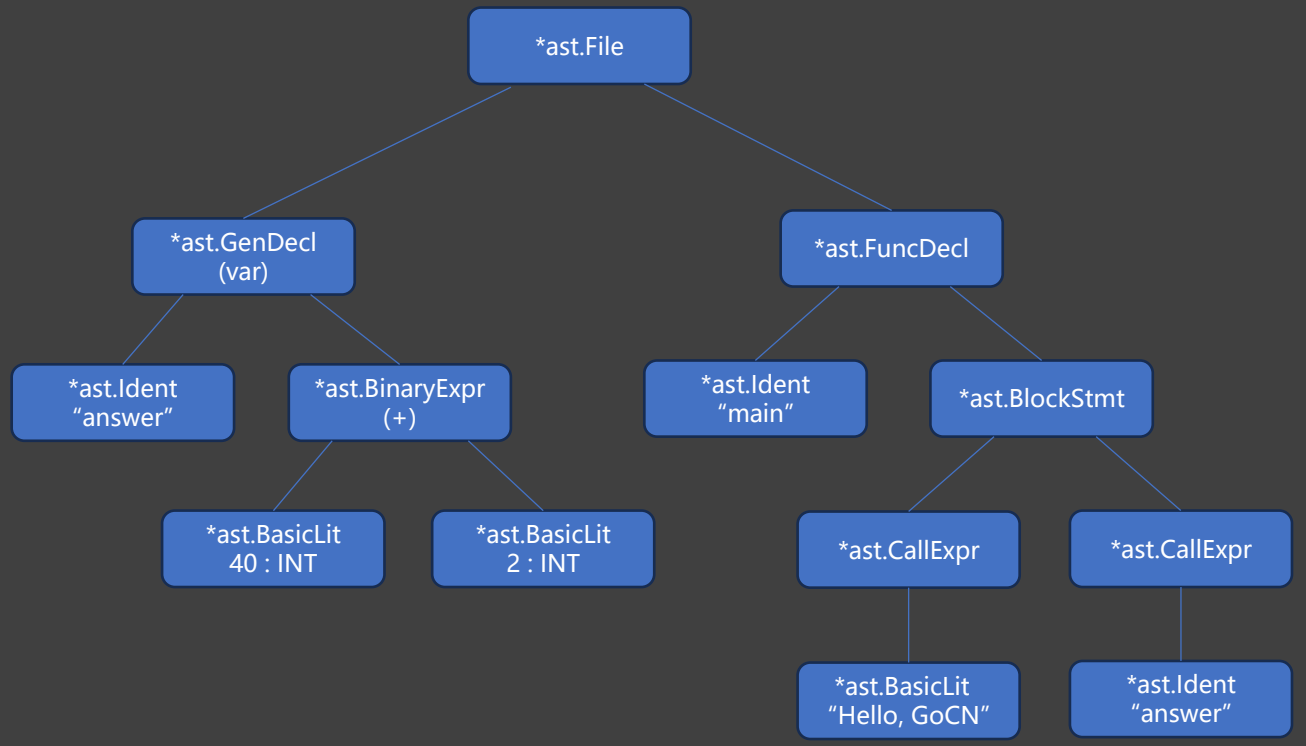
- go/token 词法单元定义
- go/scanner 词法扫描
- go/ast 抽象语法树定义
- go/parser 语法解析

Golang 编译流程简介

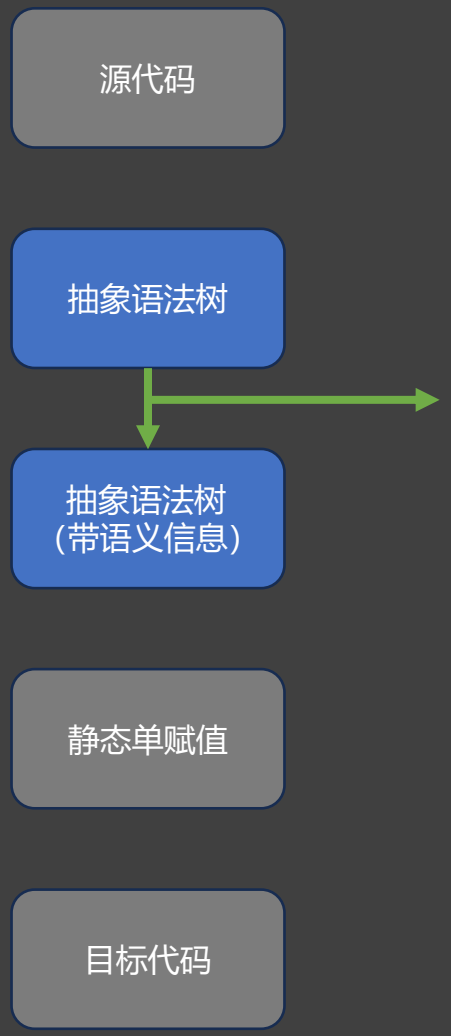


Golang 编译流程简介


```go
package main

import (
    "go/ast"
    "go/parser"
    "go/token"
)

const src = `
package main

var answer = 40 + 2

func main() {
    println("Hello, GoCN!")
    println(answer)
}
`

func main() {
    fset := token.NewFileSet()
    f, _ := parser.ParseFile(fset, "test.go", src, parser.AllErrors)
    ast.Print(nil, f)
}
```



Golang 编译流程简介



语义分析，涉及的包：

- go/types

功能：

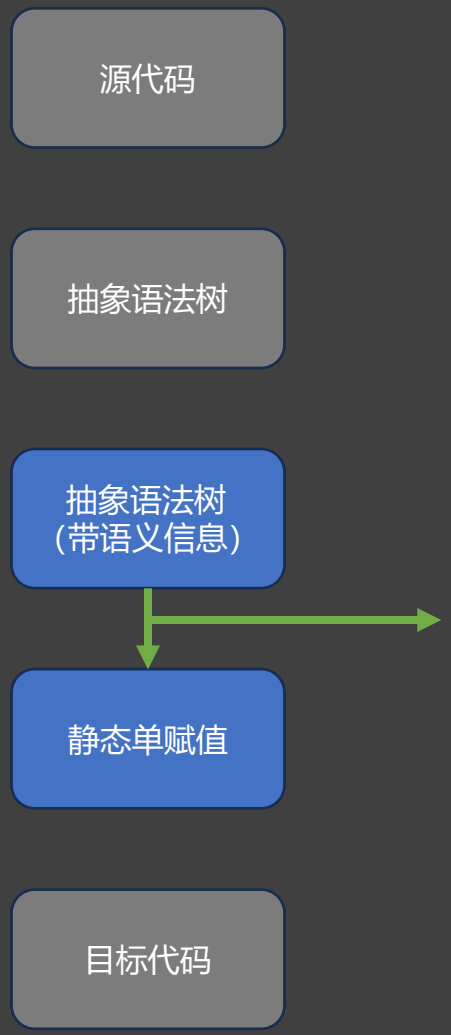
- 类型检查和推导，如：

```
v := "a" + 1
```

- 确定标识符的引用关系，如：

```
var g = 42

func main() {
    g := "Hello GoCN!"
    println(g)
}
```

Golang 编译流程简介



转静态单赋值形式，涉及的包：

- golang.org/x/tools/go/ssa

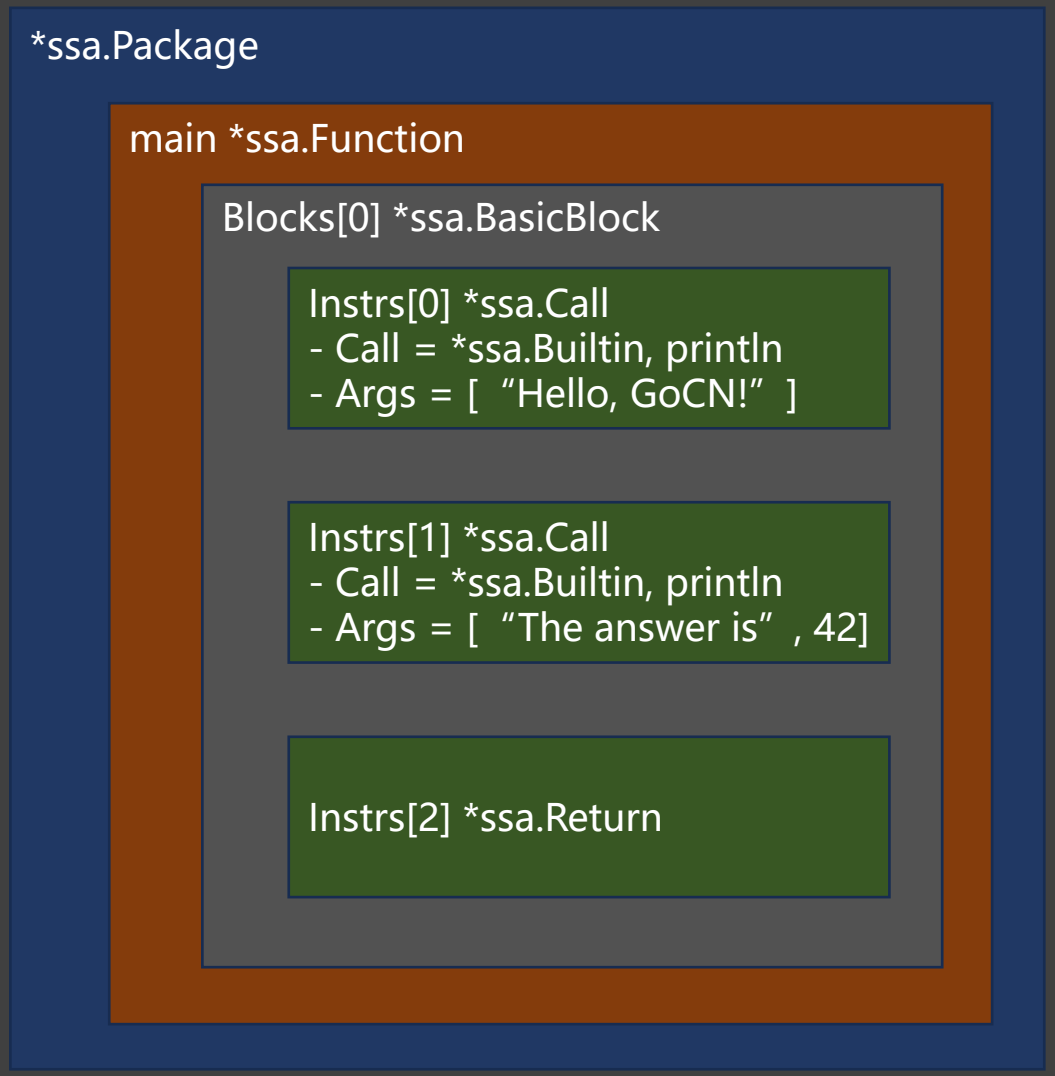
静态单赋值（Static Single Assignment，SSA），是 1988 年由 Barry K. Rosen、Mark N. Wegman、F. Kenneth Zadeck 提出的一种中间代码（IR）表示形式，典型特征是所有变量被且仅被赋值一次。

2016年，Go 1.7 加入 SSA 支持。

SSA 解释执行

```
const src = `
package main

func main() {
    println("Hello, GoCN!")
    println("The answer is:", 42)
}
`

func main() {
    …
    ssaPkg.Build()
    ssaPkg.Func("main").WriteTo(os.Stdout)
    …
}
```

```
# Package: test.go
# Location: test.go:4:6
func main():
0:                                                        entry P:0 S:0
        t0 = println("Hello, GoCN!":string)                          ()
        t1 = println("The answer is:":string, 42:int)                ()
        return
```

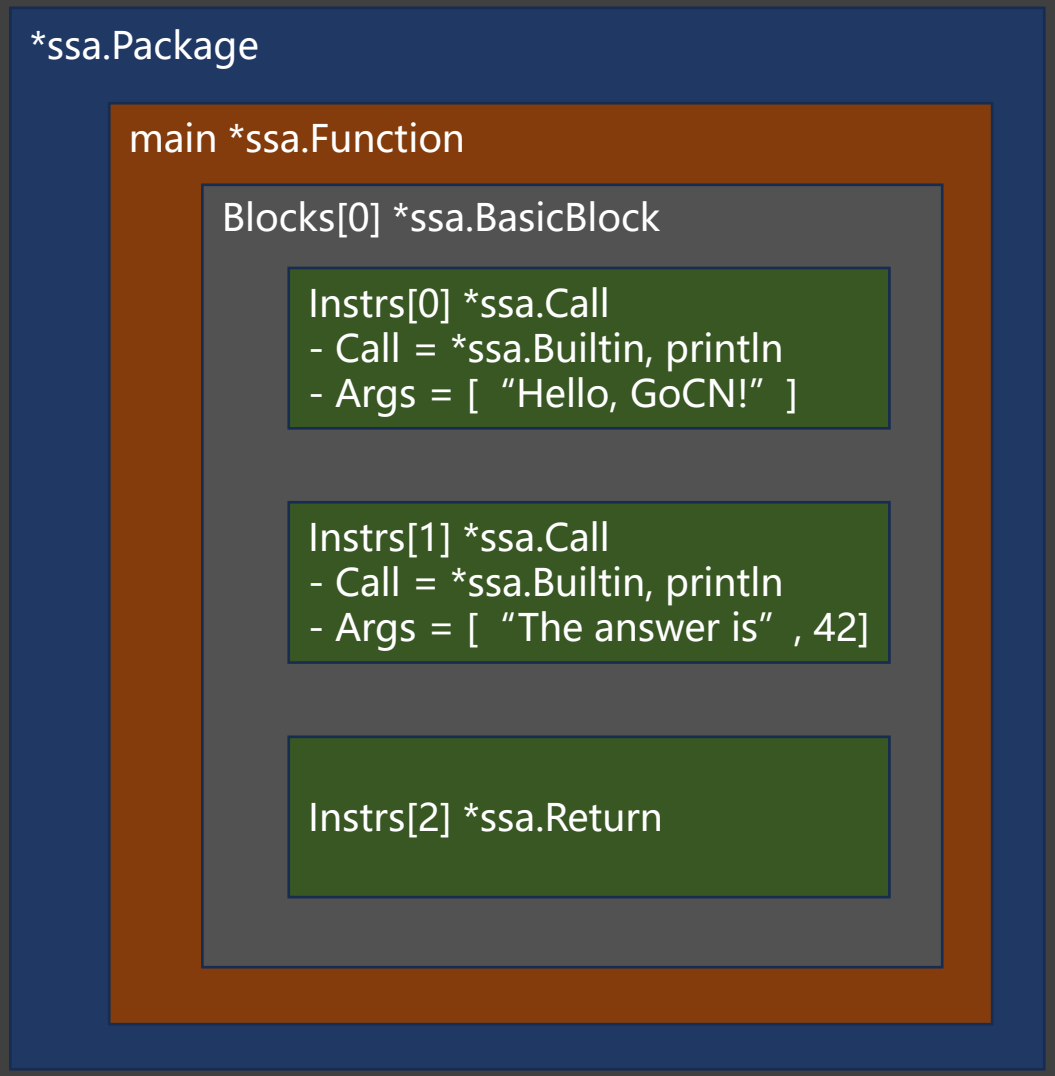


SSA 解释执行

run.go 1 ×

run.go > doCall

```
9
10      "golang.org/x/tools/go/ssa"
11  )
12
13  func runFunc(fn *ssa.Function) {
14      fmt.Println("--- runFunc begin ---")
15      defer fmt.Println("--- runFunc end   ---")
16
17      // 从第0个Block开始执行
18      if len(fn.Blocks) > 0 {
19          for blk := fn.Blocks[0]; blk != nil; {
20              blk = runFuncBlock(fn, fn.Blocks[0])
21          }
22      }
23  }
24
25  // 运行Block，返回下一个Block，如果返回nil表示结束
26  func runFuncBlock(fn *ssa.Function, block *ssa.BasicBlock) (nextBlock *ssa.BasicBlock) {
27      for _, ins := range block.Instrs {
28          switch ins := ins.(type) {
29          case *ssa.Call:
30              doCall(ins)
31          case *ssa.Return:
32              doReturn(ins)
33          default:
34              panic("Not Implemented.")
35          }
36      }
37      return nil
38  }
```

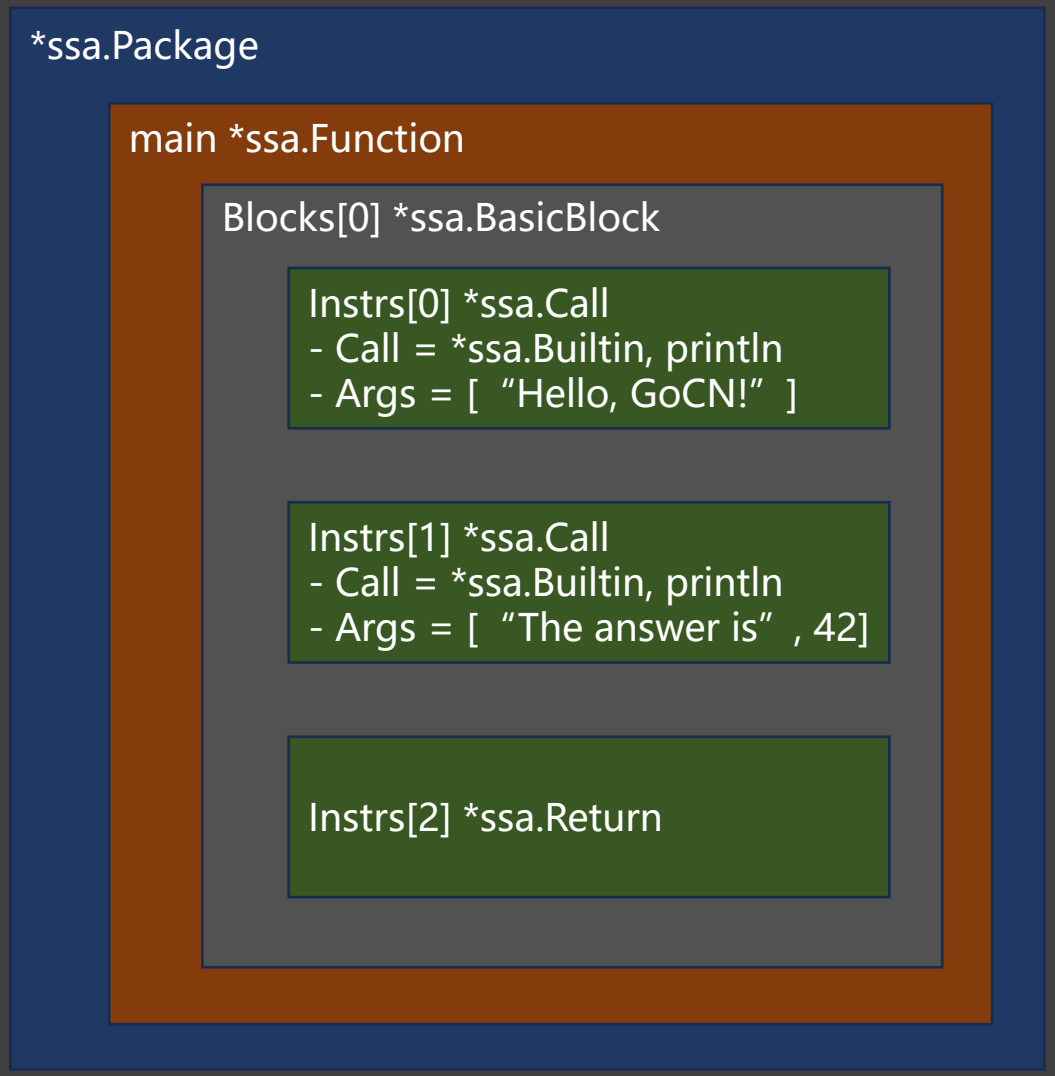


SSA 解释执行

run.go 1 ×

run.go > callBuiltin

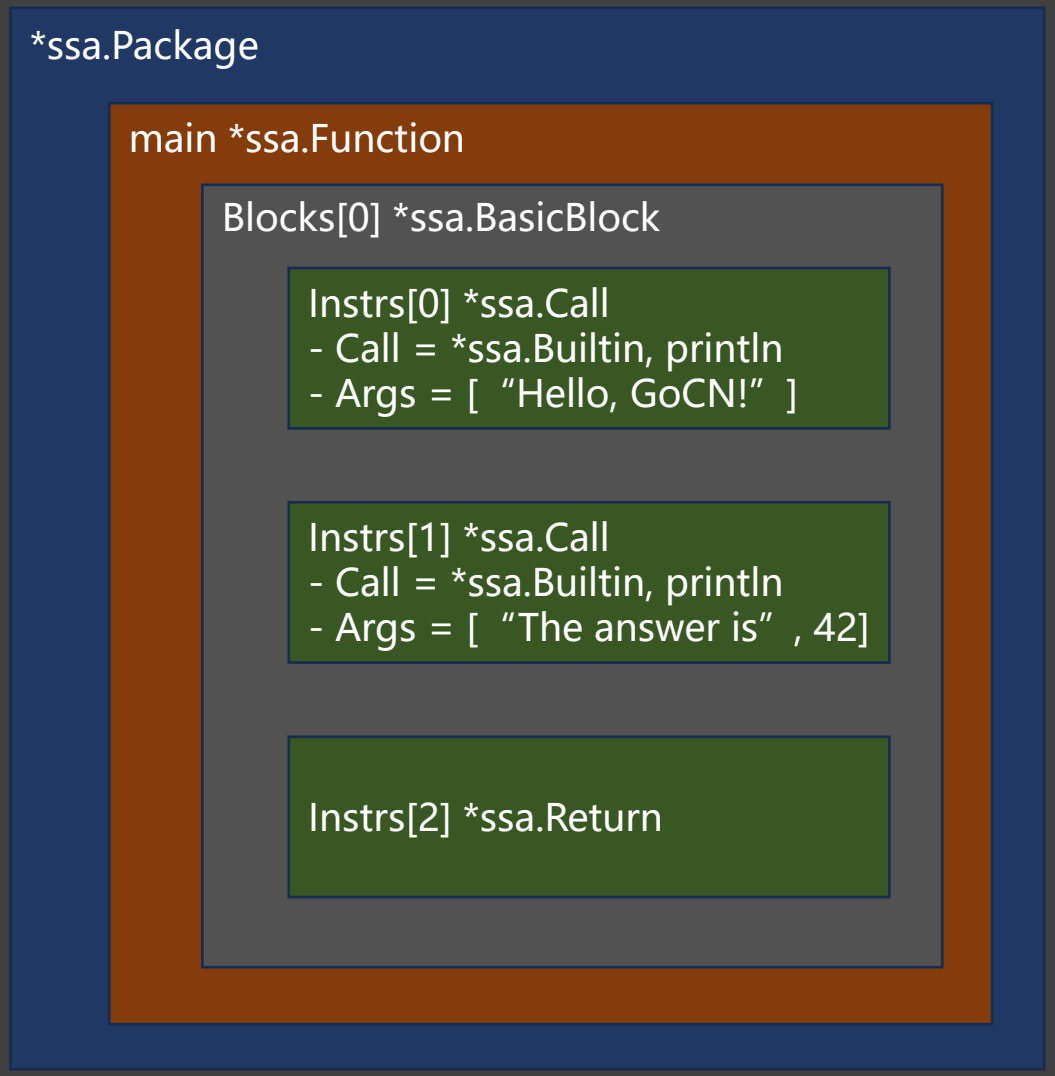
```go
// 运行Block，返回下一个Block，如果返回nil表示结束
func runFuncBlock(fn *ssa.Function, block *ssa.BasicBlock) (nextBlock *ssa.BasicBlock) {
    for _, ins := range block.Instrs {
        switch ins := ins.(type) {
        case *ssa.Call:
            doCall(ins)
        case *ssa.Return:
            doReturn(ins)
        default:
            panic("Not Implemented.")
        }
    }
    return nil
}

func doCall(ins *ssa.Call) {
    switch {
    case ins.Call.Method == nil: // 普通函数调用
        switch callFn := ins.Call.Value.(type) {
        case *ssa.Builtin:
            callBuiltin(callFn, ins.Call.Args...)
        default:
            // 普通函数
            panic("Not Implemented.")
        }

    default:
        // 方法或接口调用
        panic("Not Implemented.")
    }
}
```



SSA 解释执行

```go
func callBuiltin(fn *ssa.Builtin, args ...ssa.Value) {
    switch fn.Name() {
    case "println":
        var buf bytes.Buffer
        for i := 0; i < len(args); i++ {
            if i > 0 {
                buf.WriteRune(' ')
            }
            switch arg := args[i].(type) {
            case *ssa.Const: // 处理常量参数
                if t, ok := arg.Type().Underlying().(*types.Basic); ok {
                    switch t.Kind() {
                    case types.Int, types.UntypedInt:
                        fmt.Fprintf(&buf, "%d", int(arg.Int64()))
                    case types.String:
                        fmt.Fprintf(&buf, "%s", constant.StringVal(arg.Value))
                    default:
                        // 其它常量类型，暂不支持
                        panic("Not Implemented.")
                    }
                }
            default:
                // 暂不支持非常量参数
                panic("Not Implemented.")
            }
        }
        buf.WriteRune('\n')
        os.Stdout.Write(buf.Bytes())
```

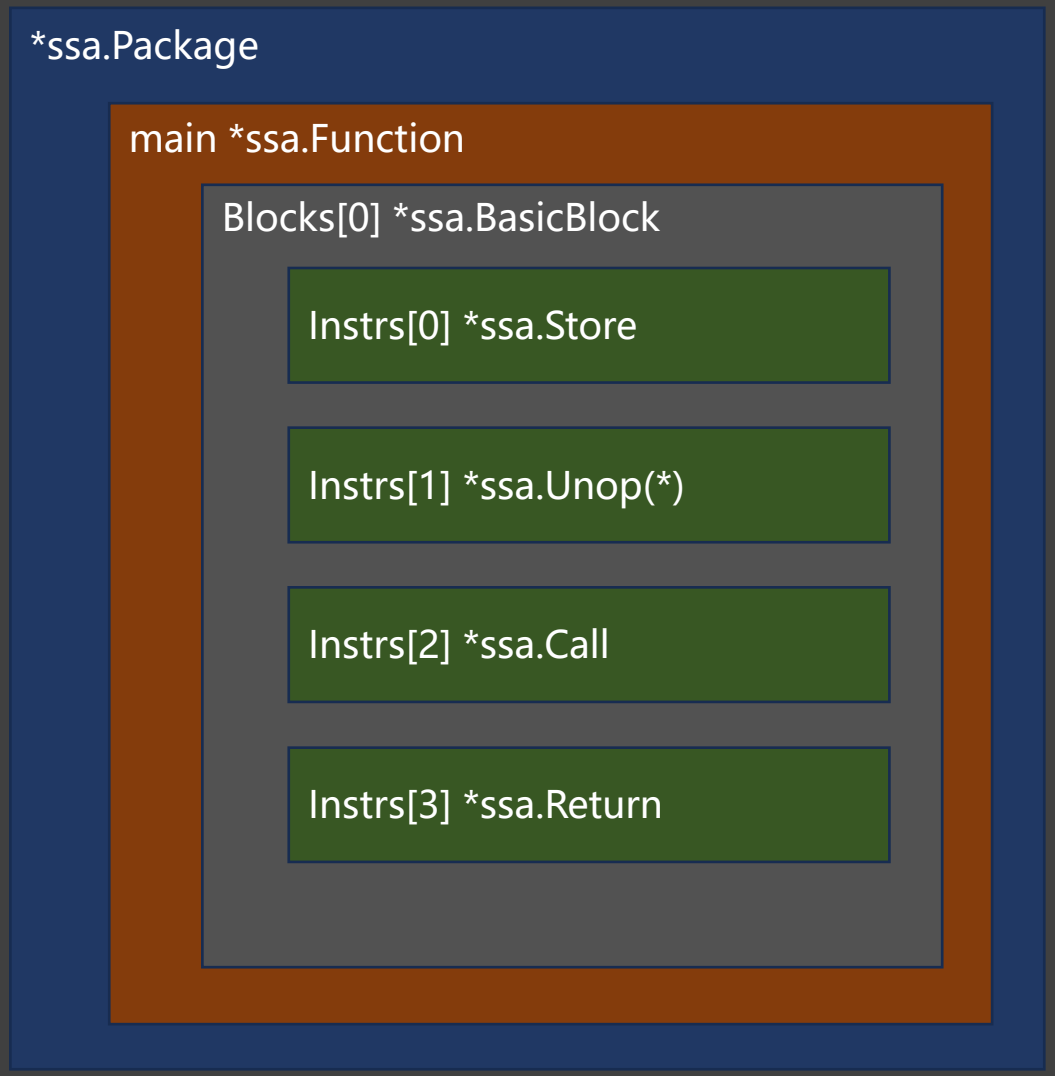
```
--- runFunc begin ---
Hello, GoCN!
The answer is: 42
--- runFunc end   ---
```



SSA 解释执行

```go
package main

var i int

func main() {
    i = 42
    println("The answer is:", i)
}
```

```
# Name: test.go.main
# Package: test.go
# Location: test.go:6:6
func main():
0:                                                entry P:0 S:0
        *i = 42:int
        t0 = *i                                           int
        t1 = println("The answer is:":string, t0)          ()
        return
```



SSA 解释执行

```go
type Engine struct {
    main     *ssa.Package
    initOnce sync.Once

    // 全局变量
    globals map[string]*watypes.Value
}

type Frame struct {
    //局部变量、虚拟寄存器等:
    env map[ssa.Value]watypes.Value
}

// 读取值(nil/全局变量/虚拟寄存器等)
func (p *Engine) getValue(fr *Frame, key ssa.Value) watypes.Value {
    switch key := key.(type) {
    case *ssa.Global:
        if r, ok := p.getGlobal(key); ok {
            return r
        }
    case *ssa.Const:
        return waops.ConstValue(key)
    case nil:
        return nil
    }

    if r, ok := fr.env[key]; ok {
        return r
    }

    panic(fmt.Sprintf("get: no value for %T: %v", key, key.Name()))
}
```

```
# Name: test.go.main
# Package: test.go
# Location: test.go:6:6
func main():
0:                                                        entry P:0 S:0
        *i = 42:int
        t0 = *i                                                    int
        t1 = println("The answer is:":string, t0)                   ()
        return
```

SSA 解释执行

```go
// 运行Block
func (p *Engine) runFuncBlock(fr *Frame, block *ssa.BasicBlock) (nextBlock *ssa.BasicBlock) {
    for _, ins := range block.Instrs {
        switch ins := ins.(type) {
        case *ssa.Store:
            println("ssa.Store")
            watypes.Store(waops.Deref(ins.Addr.Type()), p.getValue(fr, ins.Addr).(*watypes.Value), p.getValue(fr, ins.Val))

        case *ssa.UnOp:
            println("ssa.UnOp")
            fr.env[ins] = waops.UnOp(ins, p.getValue(fr, ins.X))

        case *ssa.Call:
            println("ssa.Call")
            args := p.prepareCall(fr, &ins.Call)
            fr.env[ins] = p.call(ins, args)
        }
    }
    return nil
}
```

```
--- runFunc begin ---
ssa.Store
ssa.UnOp
ssa.Call
The answer is: 42
--- runFunc end   ---
```

```
# Name: test.go.main
# Package: test.go
# Location: test.go:6:6
func main():
0:                                                                entry P:0 S:0
        *i = 42:int
        t0 = *i                                                            int
        t1 = println("The answer is:":string, t0)                           ()
        return
```

SSA 解释执行

```go
package main

var i int

func main() {
    i = 24
    println("The answer is:", i + 3 * add(2, 4))
}

func add(i int, j int) int{
    return i + j
}
```

```
package test.go:
  func  add        func(i int, j int) int
  var   i          int
  func  init       func()
  var   init$guard bool
  func  main       func()

# Name: test.go.main
# Package: test.go
# Location: test.go:10:6
func main():
0:                                                          entry P:0 S:0
        *i = 24:int
        t0 = *i                                                      int
        t1 = add(2:int, 4:int)                                       int
        t2 = 3:int * t1                                              int
        t3 = t0 + t2                                                 int
        t4 = println("The answer is:":string, t3)                     ()
        return

# Name: test.go.add
# Package: test.go
# Location: test.go:6:6
func add(i int, j int) int:
0:                                                          entry P:0 S:0
        t0 = i + j                                                   int
        return t0
```

SSA 解释执行

```go
type Frame struct {
    env       map[ssa.Value]watypes.Value //局部变量、虚拟寄存器等
    result    watypes.Value               //返回值
    block     *ssa.BasicBlock             //当前块
    prevBlock *ssa.BasicBlock             //上一个块
}

func (p *Engine) runFunc(fn watypes.Value, args []watypes.Value) watypes.Value {
    if fn, ok := fn.(*ssa.Builtin); ok {
        return callBuiltin(fn, args)
    }

    if fn, ok := fn.(*ssa.Function); ok {
        fr := NewFrame()
        fr.block = fn.Blocks[0]
        // 函数的参数添加到上下文环境
        for i, p := range fn.Params {
            fr.env[p] = args[i]
        }

        for fr.block != nil {
            p.runFrame(fr) // 核心逻辑
        }

        return fr.result
    }

    panic(fmt.Sprintf("Unknown function: %v", fn))
}
```

```
package test.go:
  func  add        func(i int, j int) int
  var   i          int
  func  init       func()
  var   init$guard bool
  func  main       func()

# Name: test.go.main
# Package: test.go
# Location: test.go:10:6
func main():
0:                                                        entry P:0 S:0
        *i = 24:int
        t0 = *i                                                    int
        t1 = add(2:int, 4:int)                                     int
        t2 = 3:int * t1                                            int
        t3 = t0 + t2                                               int
        t4 = println("The answer is:":string, t3)                   ()
        return

# Name: test.go.add
# Package: test.go
# Location: test.go:6:6
func add(i int, j int) int:
0:                                                        entry P:0 S:0
        t0 = i + j                                                 int
        return t0
```

SSA 解释执行

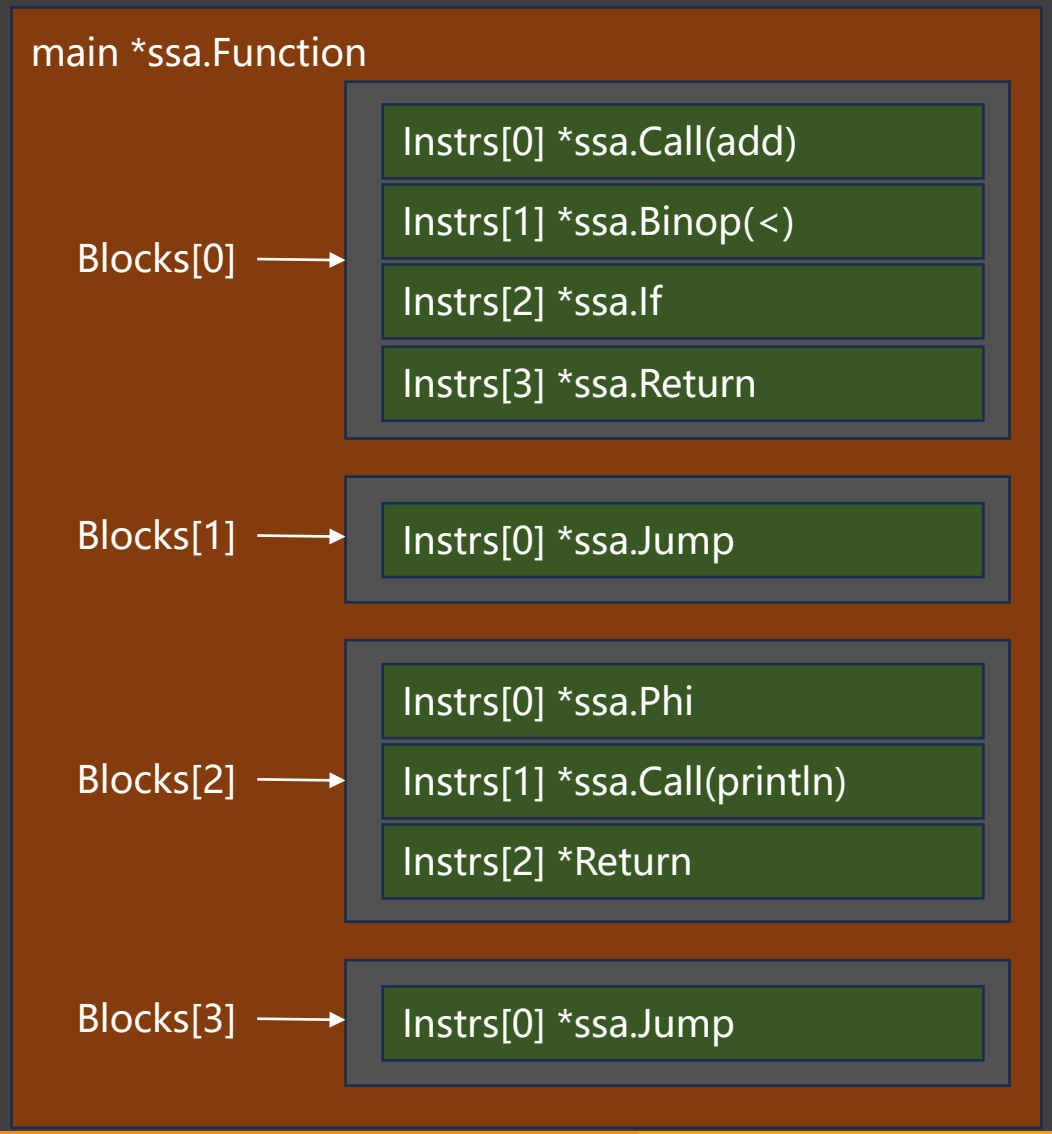
```go
func (p *Engine) runFrame(fr *Frame) {
    for i := 0; i < len(fr.block.Instrs); i++ {
        switch ins := fr.block.Instrs[i].(type) {
        case *ssa.Store: ···

        case *ssa.UnOp: ···

        case *ssa.BinOp:
            fr.env[ins] = waops.BinOp(ins.Op, ins.X.Type(),
                p.getValue(fr, ins.X), p.getValue(fr, ins.Y))

        case *ssa.Call:
            args := p.prepareCall(fr, &ins.Call)
            fr.env[ins] = p.runFunc(ins.Call.Value, args)

        case *ssa.Return:
            switch len(ins.Results) {
            case 0:
            case 1:
                fr.result = p.getValue(fr, ins.Results[0])
            default:
                panic("multi-return is not supported")
            }
            fr.block = nil
            return
        }
    }

    fr.block = nil
}
```

```
package test.go:
  func  add        func(i int, j int) int
  var   i          int
  func  init       func()
  var   init$guard bool
  func  main       func()

# Name: test.go.main
# Package: test.go
# Location: test.go:10:6
func main():
0:                                                          entry P:0 S:0
        *i = 24:int
        t0 = *i                                                      int
        t1 = add(2:int, 4:int)                                       int
        t2 = 3:int * t1                                              int
        t3 = t0 + t2                                                 int
        t4 = println("The answer is:":string, t3)                     ()
        return

# Name: test.go.add
# Package: test.go
# Location: test.go:6:6
func add(i int, j int) int:
0:                                                          entry P:0 S:0
        t0 = i + j                                                   int
        return t0
```

SSA 解释执行

```go
package main

func add(i int, j int) int{
    return i + j
}

func main() {
    var i int
    if add(3, 5) < 9{
        i = 13
    } else{
        i = 42
    }
    println(i)
}
```

```
func main():
0:                                                entry P:0 S:2
        t0 = add(3:int, 5:int)                            int
        t1 = t0 < 9:int                                  bool
        if t1 goto 1 else 3
1:                                                if.then P:1 S:1
        jump 2
2:                                                if.done P:2 S:0
        t2 = phi [1: 13:int, 3: 42:int] #i                int
        t3 = println(t2)                                   ()
        return
3:                                                if.else P:1 S:1
        jump 2
```



SSA 解释执行

```
func (p *Engine) runFrame(fr *Frame) {
    for i := 0; i < len(fr.block.Instrs); i++ {
        switch ins := fr.block.Instrs[i].(type) {
        case *ssa.If:
            if p.getValue(fr, ins.Cond).(bool) {
                //println("if:true, goto block:", fr.block.Succs[0].String())
                fr.prevBlock, fr.block = fr.block, fr.block.Succs[0] // true
            } else {
                //println("if:false, goto block:", fr.block.Succs[1].String())
                fr.prevBlock, fr.block = fr.block, fr.block.Succs[1] // false
            }
            return

        case *ssa.Jump:
            //println("jump to block:", fr.block.Succs[0].String())
            fr.prevBlock, fr.block = fr.block, fr.block.Succs[0]
            return

        case *ssa.Phi:
            for i, pred := range ins.Block().Preds {
                if fr.prevBlock == pred {
                    fr.env[ins] = p.getValue(fr, ins.Edges[i])
                    break
                }
            }
```

```
func main():
0:                                                  entry P:0 S:2
        t0 = add(3:int, 5:int)                              int
        t1 = t0 < 9:int                                    bool
        if t1 goto 1 else 3
1:                                                  if.then P:1 S:1
        jump 2
2:                                                  if.done P:2 S:0
        t2 = phi [1: 13:int, 3: 42:int] #i                  int
        t3 = println(t2)                                    ()
        return
3:                                                  if.else P:1 S:1
        jump 2
```

分支控制

SSA 解释执行

```go
package main

func add(i int, j int) int{
    return i + j
}

func fib(i0, i1, n int) (ret int) {
    print(i1, " ")
    if n <= 2 {
        ret = i0 + i1
    } else {
        ret = fib(i1, i0+i1, n-1)
    }
    return
}

func main() {
    var i int
    if add(3, 5) < 9{
        i = 13
    } else{
        i = 42
    }
    println(fib(0, 1, i))
}
```

```
# Name: test.go.main
# Package: test.go
# Location: test.go:18:6
func main():
0:                                                          entry P:0 S:2
        t0 = add(3:int, 5:int)                                      int
        t1 = t0 < 9:int                                            bool
        if t1 goto 1 else 3
1:                                                         if.then P:1 S:1
        jump 2
2:                                                         if.done P:2 S:0
        t2 = phi [1: 13:int, 3: 42:int] #i                          int
        t3 = fib(0:int, 1:int, t2)                                  int
        t4 = println(t3)                                             ()
        return
3:                                                         if.else P:1 S:1
        jump 2

# Name: test.go.fib
# Package: test.go
# Location: test.go:8:6
func fib(i0 int, i1 int, n int) (ret int):
0:                                                          entry P:0 S:2
        t0 = print(i1, " ":string)                                   ()
        t1 = n <= 2:int                                            bool
        if t1 goto 1 else 3
1:                                                         if.then P:1 S:1
        t2 = i0 + i1                                                int
        jump 2
2:                                                         if.done P:2 S:0
        t3 = phi [1: t2, 3: t6] #ret                                int
        return t3
3:                                                         if.else P:1 S:1
        t4 = i0 + i1                                                int
        t5 = n - 1:int                                              int
        t6 = fib(i1, t4, t5)                                        int
        jump 2

1 1 2 3 5 8 13 21 34 55 89 144 233
```

SSA 解释执行

ssa.Alloc	ssa.Phi	ssa.Call
ssa.BinOp	ssa.UnOp	ssa.ChangeType
ssa.Convert	ssa.ChangeInterface	ssa.MakeInterface
ssa.MakeClosure	ssa.MakeMap	ssa.MakeChan
ssa.MakeSlice	ssa.Slice	ssa.FieldAddr
ssa.Field	ssa.IndexAddr	ssa.Index
ssa.Lookup	ssa.Select	ssa.Range
ssa.Next	ssa.TypeAssert	ssa.Extract
ssa.Jump	ssa.If	ssa.Return
ssa.RunDefers	ssa.Panic	ssa.Go
ssa.Defer	ssa.Send	ssa.Store
ssa.MapUpdate		

基于 SSA 的应用

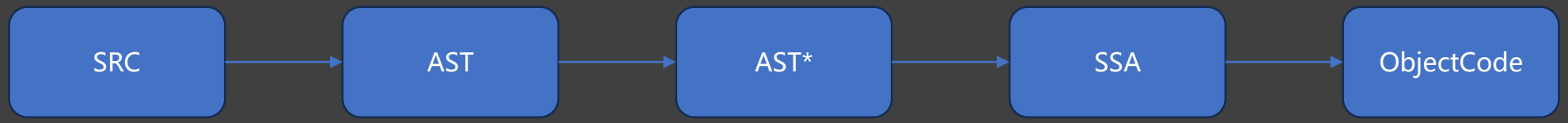
脚本解释器

基于 SSA 的应用

```
const src = `
package main

func main() {
    println("Hello, GoCN!")
    println("The answer is:", 42)
}
`

func main() {
    …
}
```

```
int main() {
        printf("%s\n", "Hello, GoCN!");
        printf("%s%d\n", "The answer is:", 42);
        return 0;
}
```

```
func runFunc(fn *ssa.Function) {
    fmt.Printf("int %s() {\n", fn.Name())
    defer fmt.Println("\treturn 0;\n}")

    // 从第0个Block开始执行
    if len(fn.Blocks) > 0 {…
    }
}

func callBuiltin(fn *ssa.Builtin, args ...ssa.Value) {
    switch fn.Name() {
    case "println":
        var format, data bytes.Buffer
        format.WriteRune('"')
        for i := 0; i < len(args); i++ {
            data.WriteString(", ")
            switch arg := args[i].(type) {
            case *ssa.Const: // 处理常量参数
                if t, ok := arg.Type().Underlying().(*types.Basic); ok {
                    switch t.Kind() {
                    case types.Int, types.UntypedInt:
                        format.WriteString("%d")
                        fmt.Fprintf(&data, "%d", int(arg.Int64()))
                    case types.String:
                        format.WriteString("%s")
                        fmt.Fprintf(&data, "\"%s\"", constant.StringVal(arg.Value))
                    default:
                        // 其它常量类型，暂不支持
                        panic("Not Implemented.")
                    }
                }
            default:
                // 暂不支持非常量参数
                panic("Not Implemented.")
            }
        }
        format.WriteString("\\n\"")
        fmt.Printf("\tprintf(%s%s);\n", format.String(), data.String())
```


SRC
AST
AST*
SSA
ObjectCode

“

谢谢！