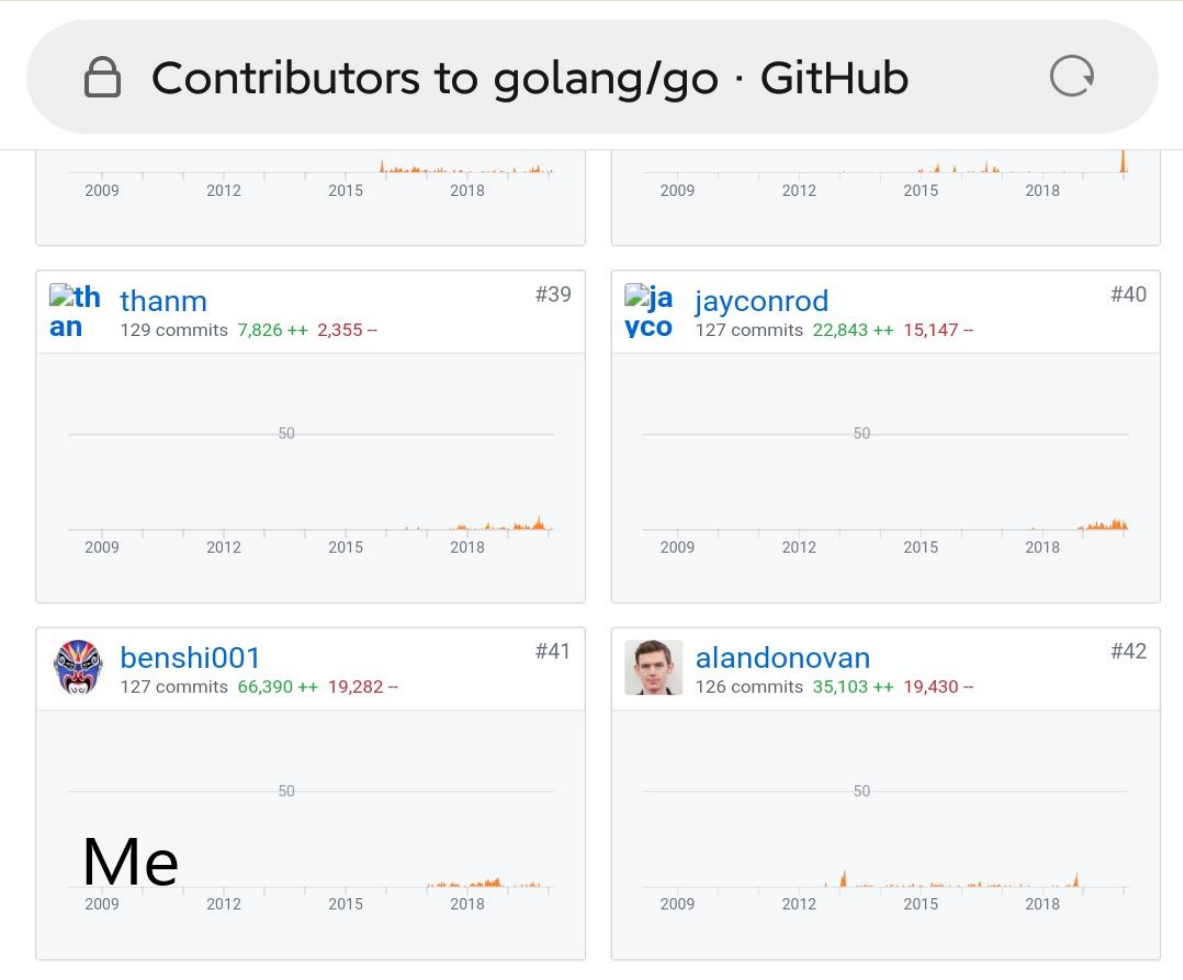

GOPHER CHINA 2020
中国 上海 / 2020-11.21-22
Go语言编译器简介
史斌

关于我



给Go编译器提交过127个补丁，累计六万余行；

拥有Go官方git仓库提交权限；

全球贡献者排名长期处于前50名；

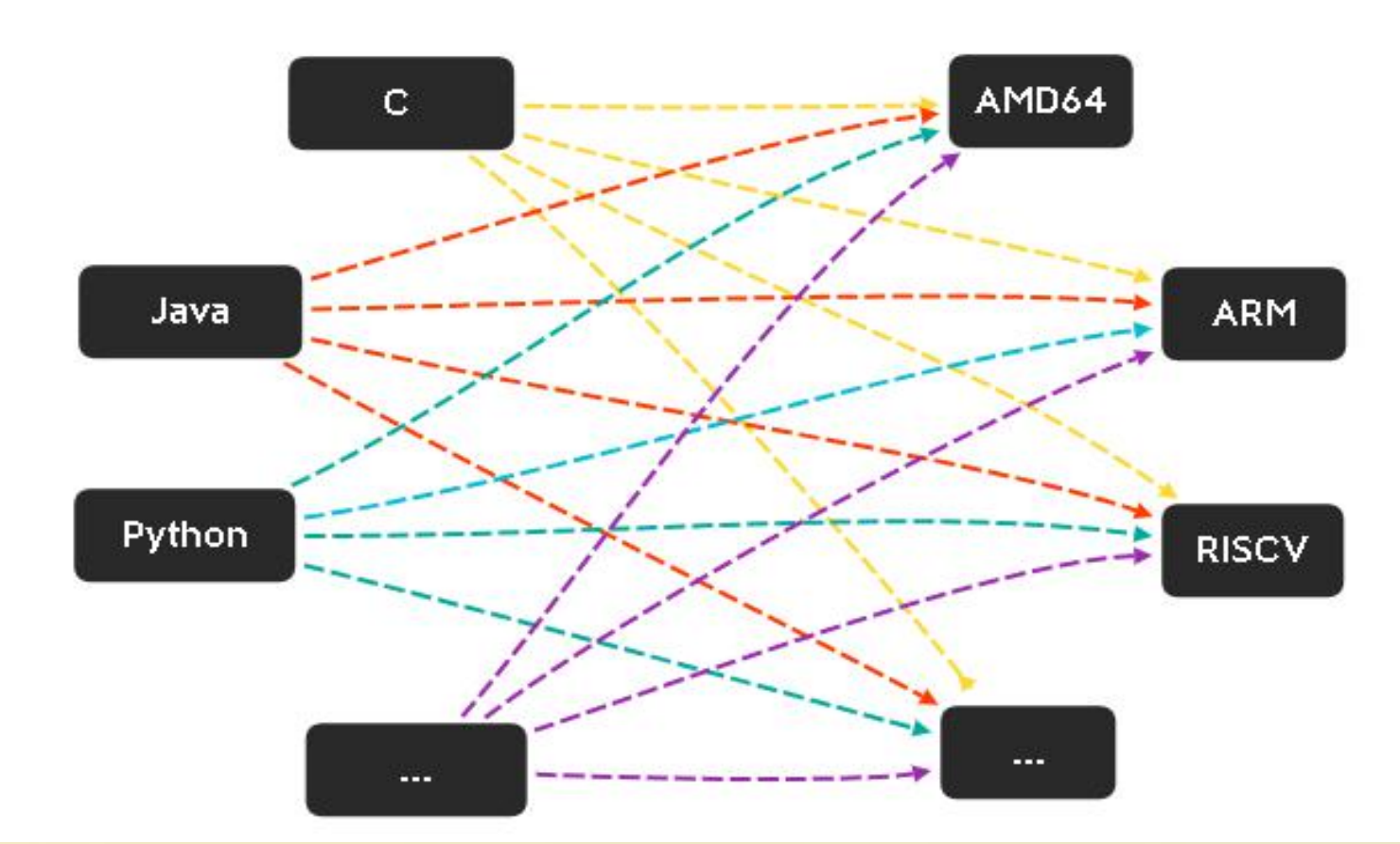
世界上90%的gopher都用过我写的代码；

编译器的重要性

- 只有1%的程序员懂汇编语言
- 汇编语言无法构建大型系统
- 操作系统内核也需要编译器才能运行起来
- 编译理论是图灵奖大户，仅次于计算复杂度理论
- 操作系统有后门，编译器的后门更致命

编译器的难题：任务爆炸

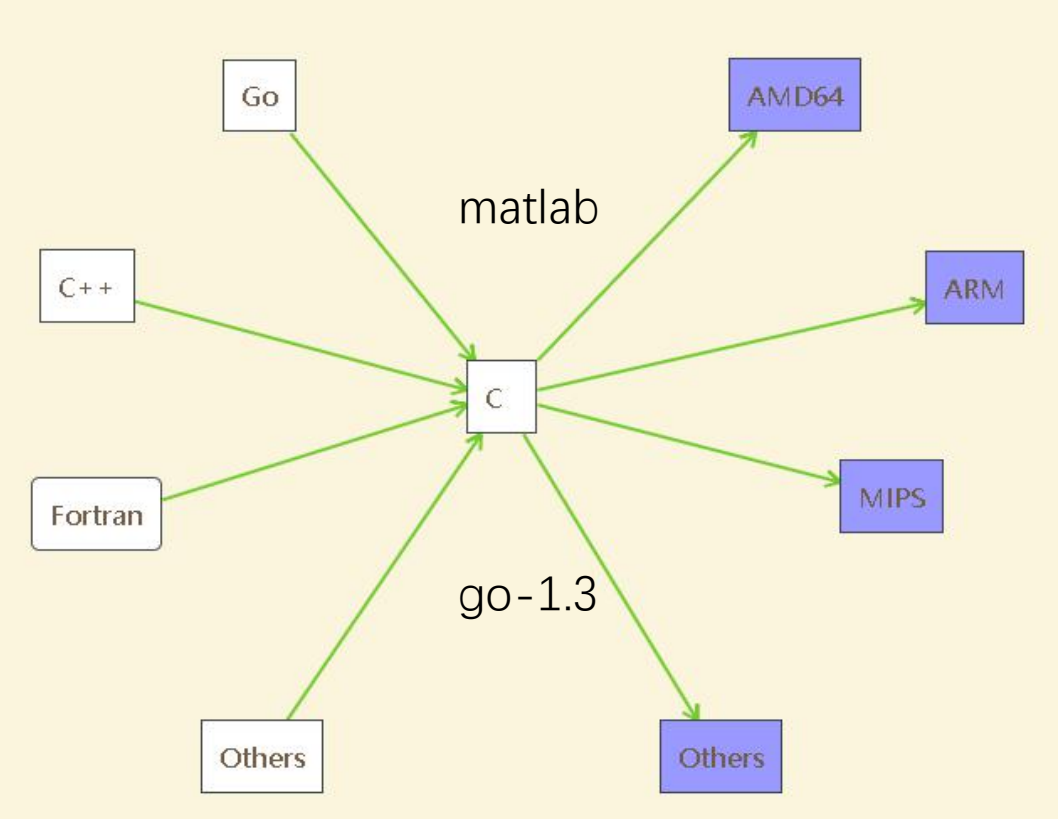
N种语言 * M种机器 = N*M 个任务



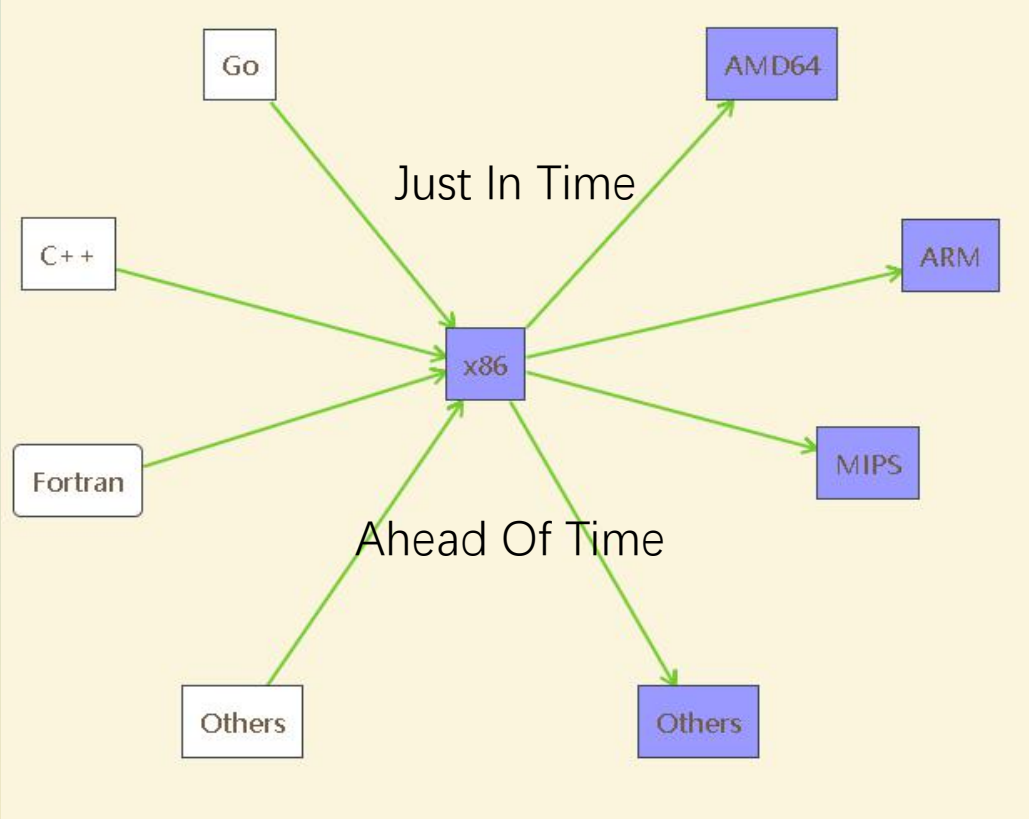
两个方案

N种语言 + M种机器 = N+M 个任务

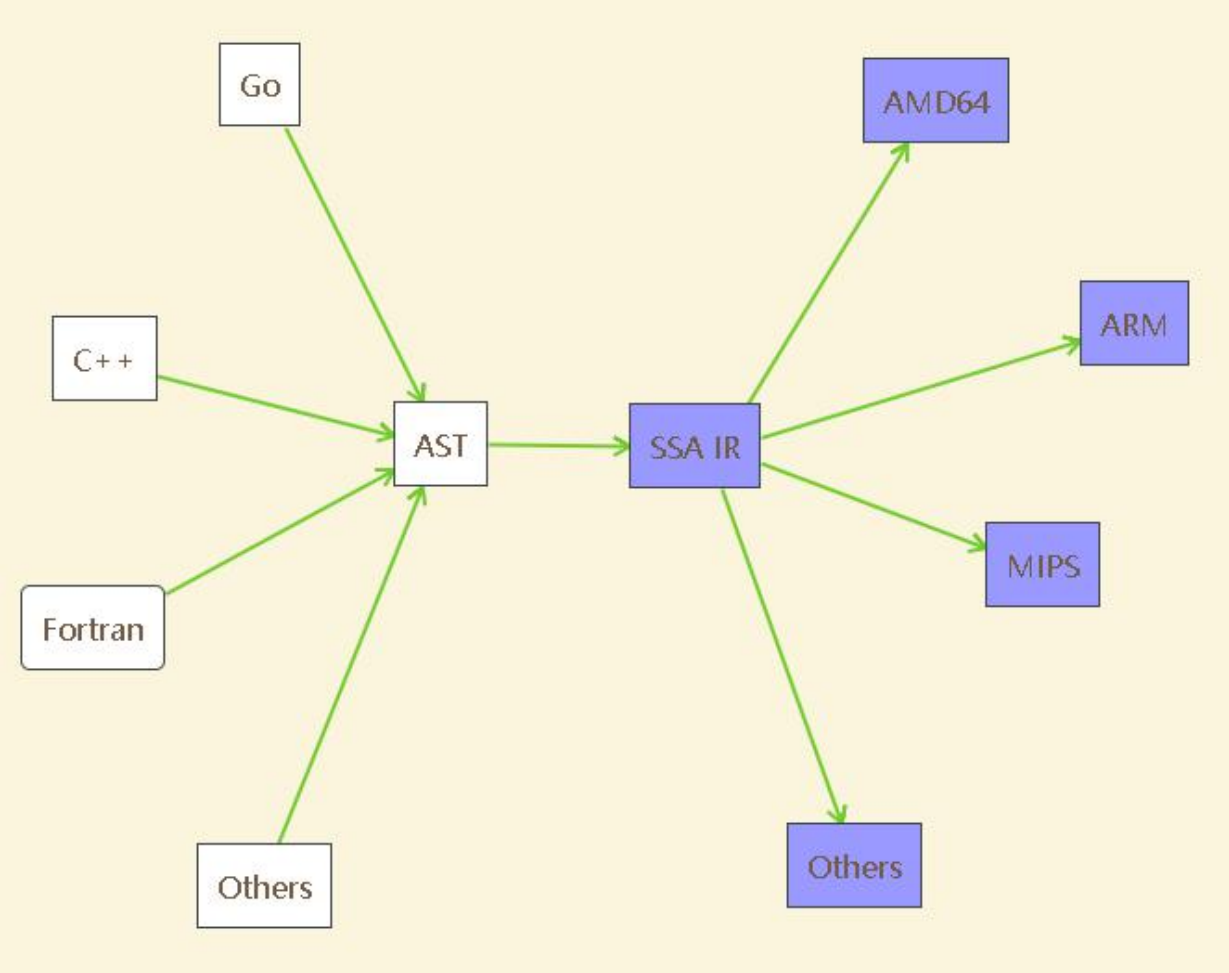
其它语言 -> C -> 各个机器



各个语言 -> x86 -> 其它机器



通用（非专用）编译器的方案



AST = Abstract Syntax Tree
抽象语法树

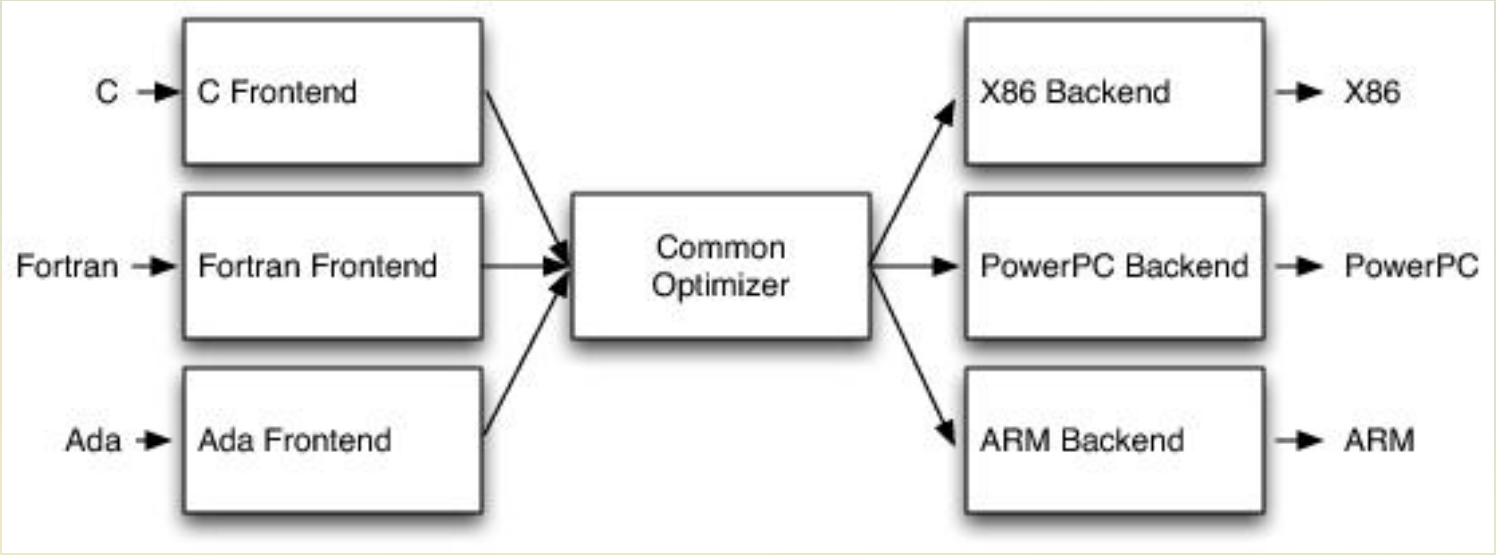
SSA = Single Static Assignment
单静态赋值

IR = Intermediate Representation
中间表示

优点:
1. 减少任务;
2. 代码复用;
3. 相关理论成熟高效;

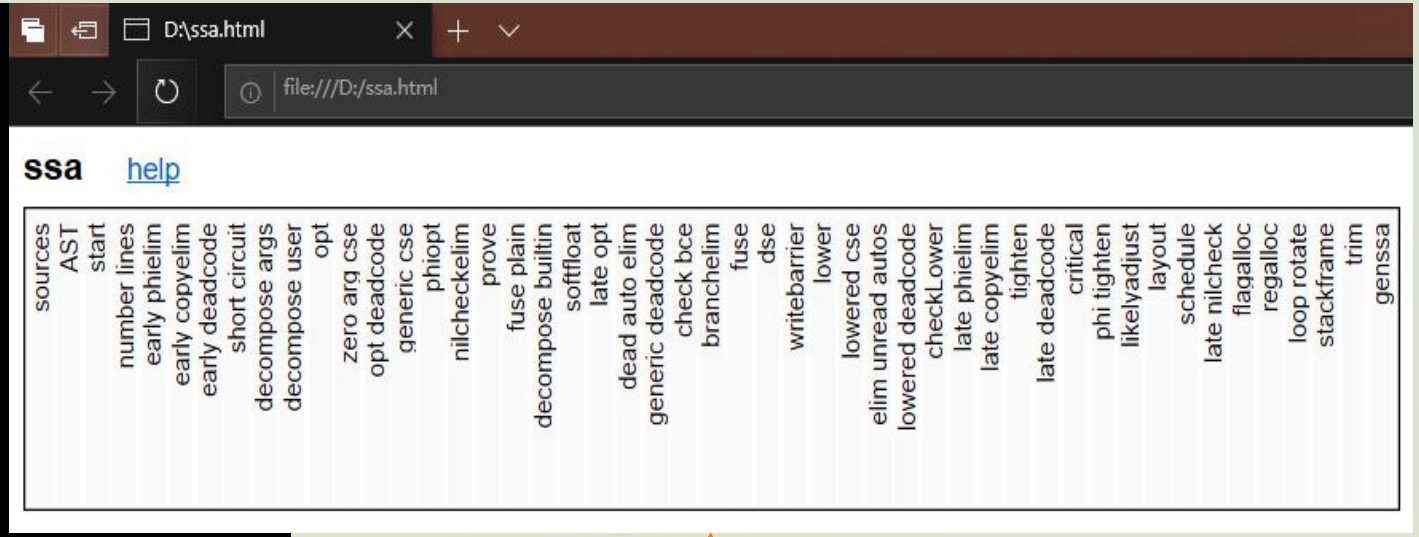
LLVM的三阶段结构

前端 中端 后端



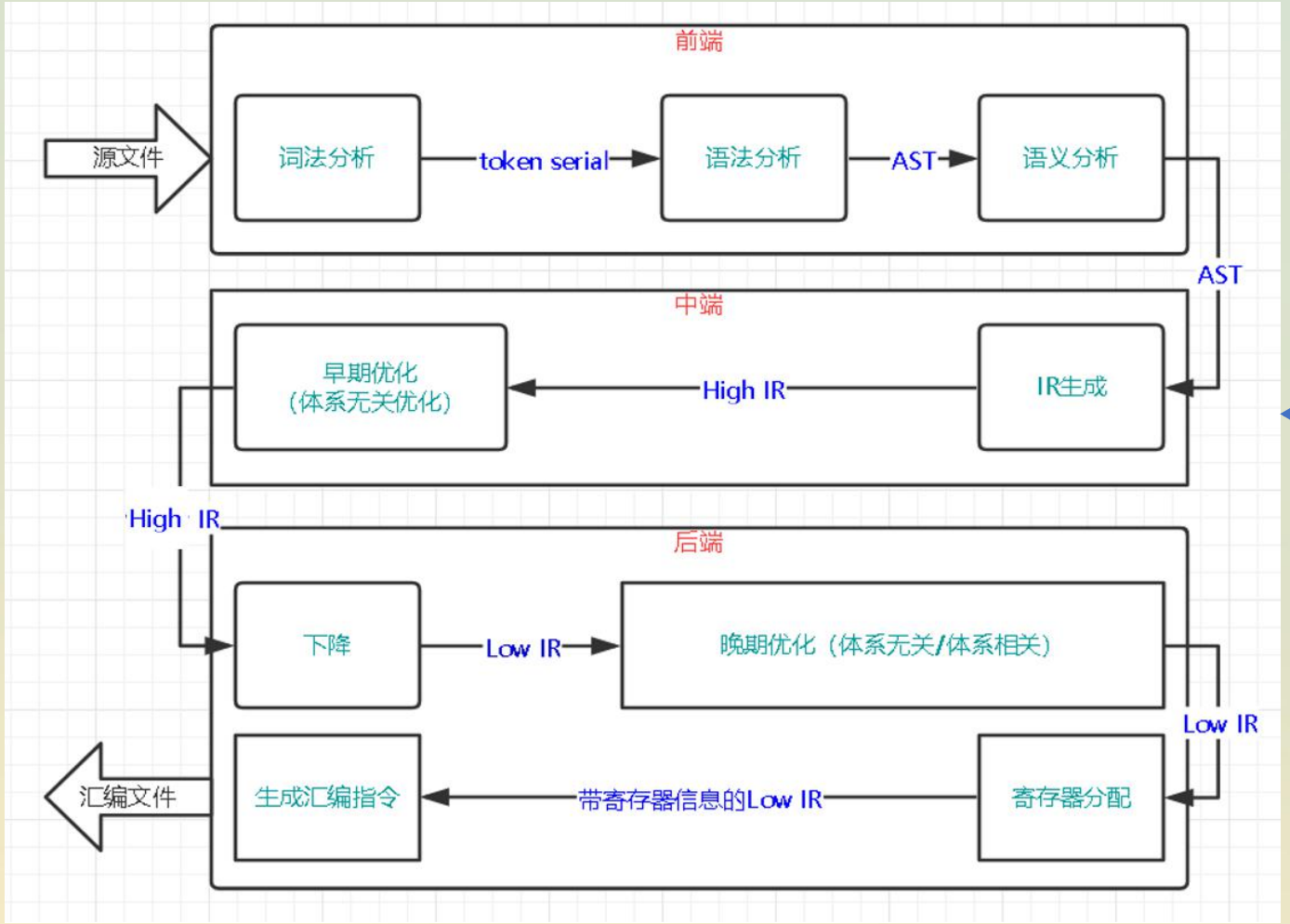
Go编译器工作流程初窥

```
ben.shi@BJDT046:/tmp$ cat a.go
package main

import "fmt"

func ssa(a, b, c uint) uint {
        return a*b + c
}

func main() {
        fmt.Println(ssa(1, 2, 3))
}
ben.shi@BJDT046:/tmp$ GOSSAFUNC=ssa go build a.go
# command-line-arguments
dumped SSA to ./ssa.html
ben.shi@BJDT046:/tmp$
```

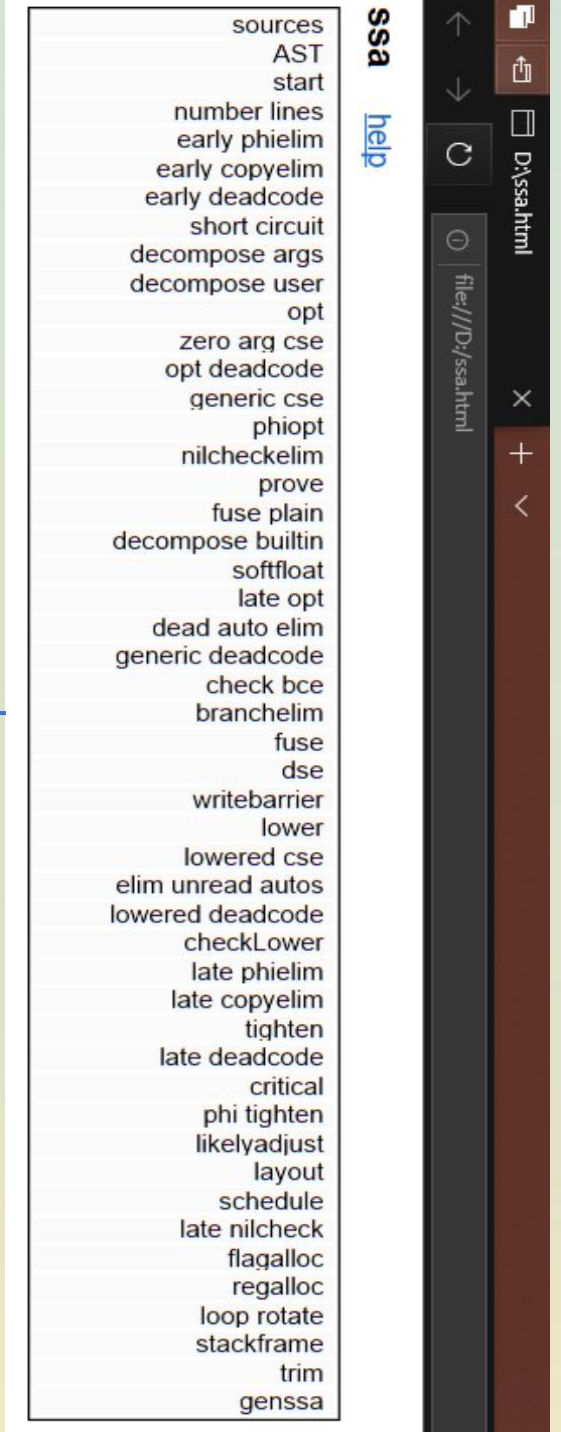


从源码到汇编，需要48道工序！

Go编译器工作流程

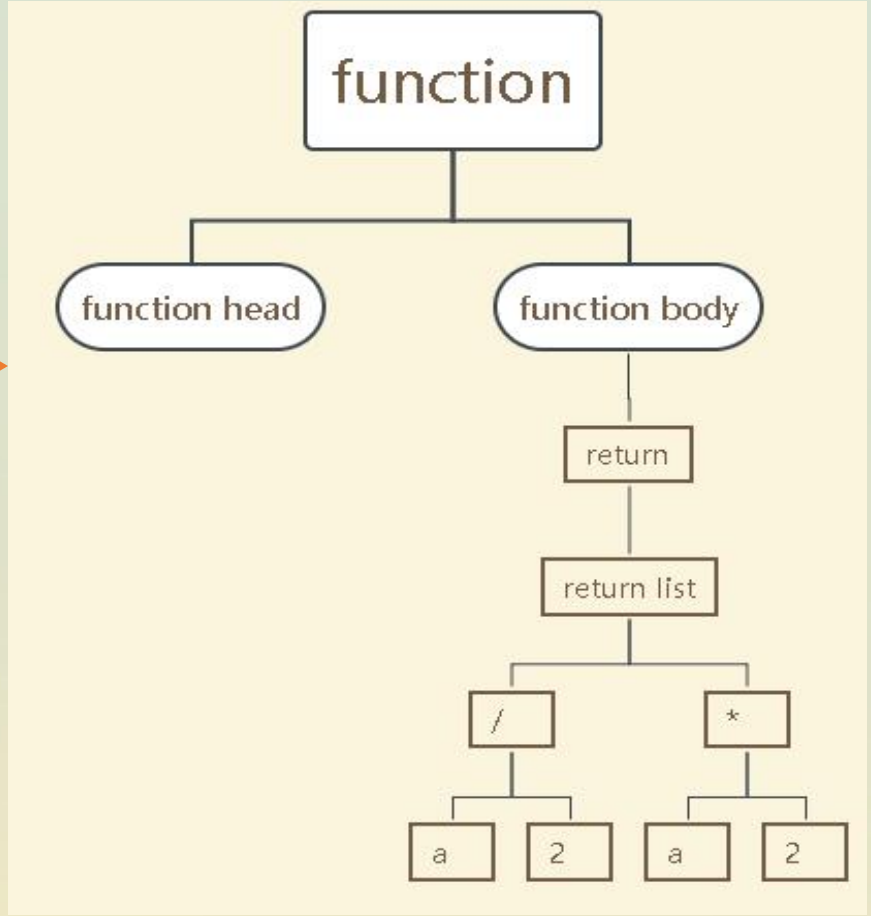


48个工序，被分为九个阶段，前中后三端。



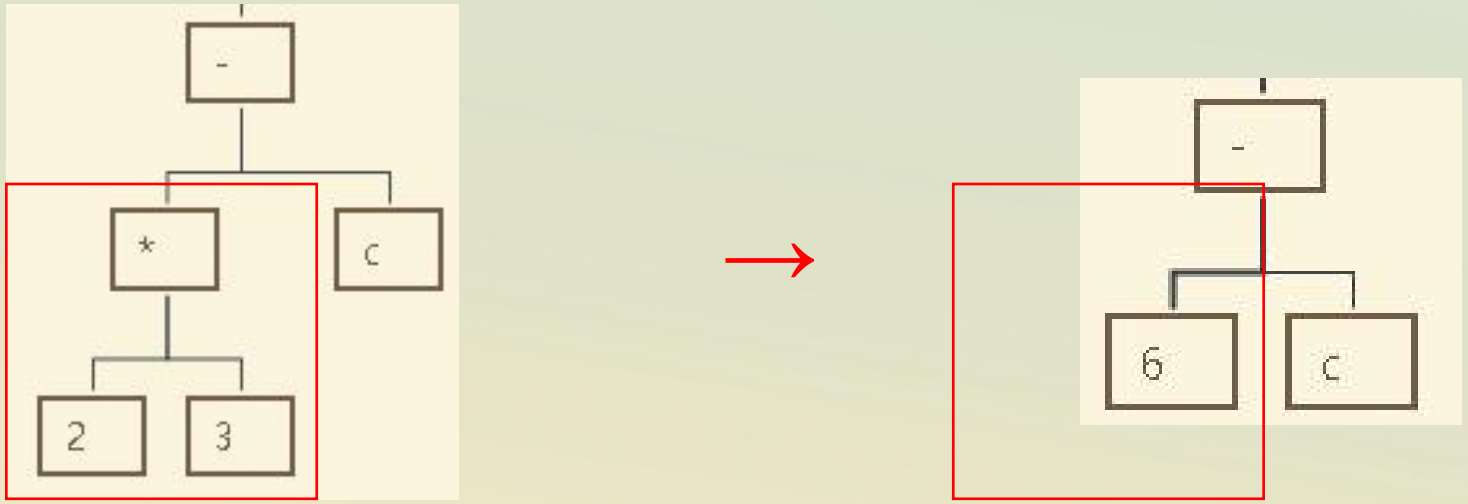
前端语法分析：Source -> AST

```
func ssa(a int) (int, int) {
        return a / 2, a * 2
}
```



前端语义分析：常量折叠

a = 2 * 3 - c → a = 6 - c

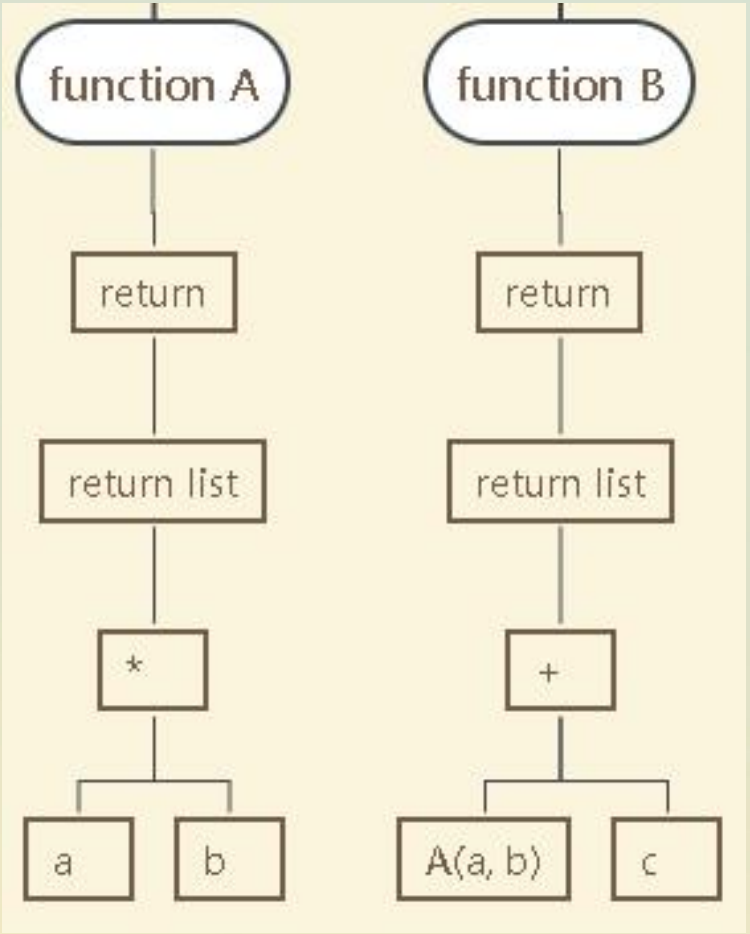


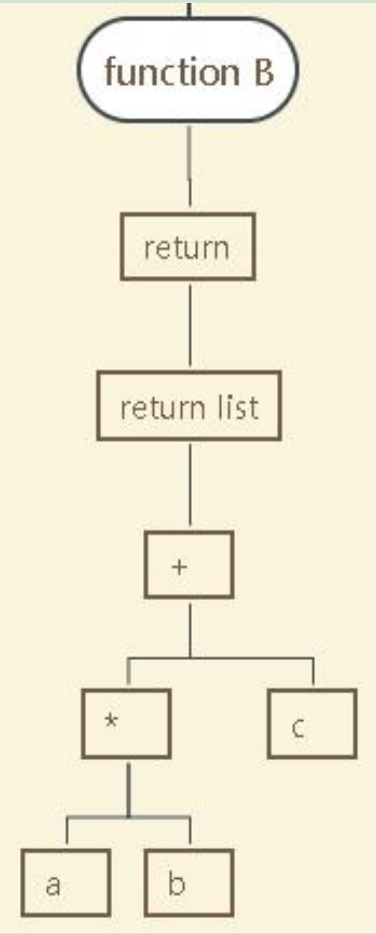
前端语义分析：内联

```
func A(a b int) int {
        return a * b
}
func B(a, b, c int) int {
        return A(a, b) + c
}
```

↓

```
func B(a, b, c int) int {
        return a*b + c
}
```



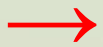


前端语义分析

- 类型检查
- 确定变量的作用域
- 内联
- 常量折叠/常量传播
- 闭包分析
- 逃逸分析
- 导入（import/include）
- 其它

SSA IR

```
func ssa(n uint) uint {
        s := uint(0)
        for i := 0; i < n; i++ {
                s = s + i * i
        }
        return s
}
```



```
ssa:
        n := param[0]
        s := 0
        i := 0
_loop:
        c := (i >= n)
        if (c) goto _end
        tmp0 := i * i
        s = s + tmp0
        i = i + 1
        goto _loop
_end:
        ret[0] = s
```

SSA IR: 没有人类可读的结构，
它更像汇编语言：

1. **赋值**
2. **一元/二元运算**
3. **goto**
4. **if-goto**
5. **传递参数/返回值**

SSA IR

```
func ssa(n, m int) int {
        if n > m {
                return n - m
        } else {
                return m - n
        }
}
```

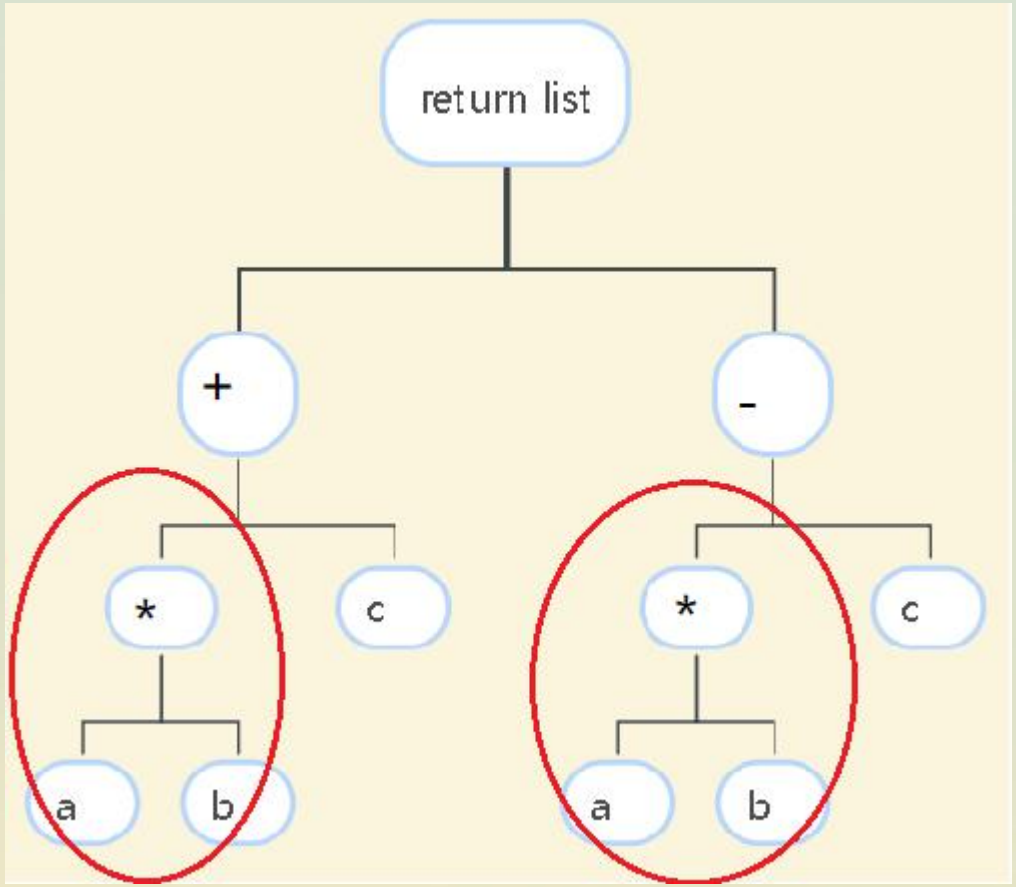
```
ssa:
        n := param[0]
        m := param[1]
        c := (n <= m)
        var tmp0
        if (c) goto _b2
        tmp0 = n - m
        goto _end
_b2:
        tmp0 = m - n
_end:
        ret[0] = tmp0
```

SSA IR: 没有人类可读的结构，
它更像汇编语言：

1. **赋值**
2. **一元/二元运算**
3. **goto**
4. **if-goto**
5. **传递参数/返回值**

中端体系无关优化：公共子表达式消除

```
//go:noinline
func ssa(a, b, c int) (int, int) {
        return a*b + c, a*b -c
/*      d := a * b
        return d + c, d - c
*/
}
```



tmp0 = a * b
tmp1 = tmp0 + c
tmp3 = **tmp0** - c
ret[0] = tmp1
ret[1] = tmp3

tmp0 = a * b
tmp1 = tmp0 + c
tmp2 = a * b
tmp3 = **tmp2** - c
ret[0] = tmp1
ret[1] = tmp3

中端体系无关优化：运算强度消减

```
//go:noinline
func ssa(a uint) uint {
        return a * 4
//      return a << 2
}
```

→

ssa:

a = param[0]
tmp0 = a * 4
ret[0] = tmp0

→

ssa:

a = param[0]
tmp0 = a << 2
ret[0] = tmp0

```
//go:noinline
func ssa(a uint) uint {
        return a % 256
}
```

→

ssa:

a = param[0]
tmp0 = a % 256
ret[0] = tmp0

→

ssa:

a = param[0]
tmp0 = a & 255
ret[0] = tmp0

中端体系无关优化：常量折叠

```
//go:noinline
func ssa(a uint) (uint) {
        b := a * 3
        return b * 3
//      return a * 9
}
```

ssa:

a = param[0]
b = a * 3
tmp0 = b * 3
ret[0] = tmp0

ssa:

a = param[0]
tmp0 = a * 9
ret[0] = tmp0

中端体系无关优化：循环优化

```
//go:noinline
func ssa(a, b int) int {
        c := int(0)
        for i := 0; i < 10; i++ {
                d := a + b
                c += d
        }
/*
        d := a + b
        for i := 0; i < 10; i++ {
                c += d
        }
*/
        return c
}
```

循环不变量放到循环体外面

Go编译器尚未实现！

```
//go:noinline
func ssa(a, b int) int {
        c := uint(0)
        for i := 0; i < 10; i++ {
                d := b + a*i
                c += d
        }
/*
        d := b
        for i := 0; i < 10; i++ {
                c += d
                d += a
        }
*/
        return c
}
```

循环归纳变量：变乘为加

中端体系无关优化

- 常量传播/常量折叠
- 复制传播
- 强度消减
- 循环不变量/归纳变量
- 冗余消除
- 活跃分析
- 向量优化
- 其它

后端：指令选择

正则表达式模式匹配

```
// Lowering arithmetic
(Add(Ptr|32|16|8) x y) -> (ADDL  x y)
(Add(32|64)F x y) -> (ADDS(S|D) x y)
(Add32carry x y) -> (ADDLcarry x y)
(Add32withcarry x y c) -> (ADCL x y c)

(Sub(Ptr|32|16|8) x y) -> (SUBL  x y)
(Sub(32|64)F x y) -> (SUBS(S|D) x y)
(Sub32carry x y) -> (SUBLcarry x y)
(Sub32withcarry x y c) -> (SBBL x y c)

(Mul(32|16|8)  x y) -> (MULL  x y)
(Mul(32|64)F x y) -> (MULS(S|D) x y)
(Mul32uhilo x y) -> (MULLQU x y)

(Select0 (Mul32uover x y)) -> (Select0 <typ.UInt32> (MULLU x y))
(Select1 (Mul32uover x y)) -> (SETO (Select1 <types.TypeFlags> (MULLU x y)))
```

后端：X86相关优化

```
x=0                  // 优化成x=x^x，因为"XOR AX,AX"的指令码比"MOV $0,AX"更短
x=y*2+z              // 使用LEA指令，避免先移位后相加两步运算
x=x&0xffff7fff       // 使用BCLR指令将bit-15清零（其它位不变），取消位与运算
x++                  // 使用"INC AX"取代"ADD $1,AX"
x--                  // 使用"DEC AX"取代"SUB $1,AX"
if ((x&0x8)==0)      // 使用TEST指令，避免AND+CMP两步运算
if (a == 0)          // 使用"TEST AX,AX"，避免"CMP $0,AX"
a > 0 ? b : c        // 使用CMOV，避免分支跳转
if                   // 使用JZ/JC/JNZ等条件跳转指令
```

后端：ARM相关优化

```
x=y+z*8              // 加/与/或运算指令，其中一个操作数可以带移位，单周期完成
x=y-z*8              // SUB指令，减数可以带移位，单周期完成
x=z*8-y              // RSB指令，被减数可以带移位，单周期完成
d=x*y+z              // 使用单条MLA指令，避免先乘后加两步操作
d=z-x*y              // 使用单条MLS指令，避免先乘后减两步操作
x=y+0xffff           // 分解成两步，x=y+0xff，x=x+0xff00，优化常量池
x=y+0xaeff80         // 分解成两步，x=y+0xaf0000，x=x-0x80，优化常量池
if ((x&0x8)==0)      // 使用TST指令，避免AND+CMP两步运算
if ((x^0x8)==0)      // 使用TEQ指令，避免XOR+CMP两步运算
if ((x+y)>0)         // 使用CMN指令，避免ADD+CMP两步运算
```

后端机器相关优化

- 高效指令选择
- 指令调度
- 流水线优化
- 缓存优化
- 并行优化
- 窥孔优化
- 其它

我对Go编译器的优化

cmd/compile: optimize ARM64's code with MADD/MSUB

MADD does MUL-ADD in a single instruction, and MSUB does the similiar simplification for MUL-SUB.

The CL implements the optimization with MADD/MSUB.

1. The total size of pkg/android_arm64/ decreases about 20KB, excluding cmd/compile/.

2. The go1 benchmark shows a little improvement for RegexpMatchHard_32-4 and Template-4, excluding noise.

t0 = b * c : **MUL Rb, Rc, Rt0**
t1 = a + t0 : **ADD Ra, Rt0, Rt1**

t1 = a + b * c : **MADD Rb, Rc, Ra, Rt1**

普通的机器指令只有两个操作数并生成一个结果，因此a+b*c需要拆分成MUL/ADD两条指令；而ARM64上有MADD指令，带三个操作数，一次生成a+b*c的结果。此类优化并不适用X86。

https://github.com/golang/go/commits?author=benshi001

我对Go编译器的优化

cmd/compile: optimize arm64 with indexed FP load/store

The FP load/store on arm64 have register indexed forms. And this CL implements this optimization.

1. The total size of pkg/android_arm64 (excluding cmd/compile) decreases about 400 bytes.

2. There is no regression in the go1 benchmark, the test case GobEncode even gets slight improvement, excluding noise.

var array float32[]
address = base + index : **ADD Rb, Ri, Ra**
temp = *addr : **FMOVS (Ra), Fx**

var array float32[]
temp += *(base + index) : **FMOVS (Rb)(Ri), Fx**

从数组中取一个浮点数，需要先计算地址（数组基地址加索引），然后读取内存，共两条指令；而ARM64读取内存指令可以同时包括基地址和索引偏移量。

https://github.com/golang/go/commits?author=benshi001

我对Go运行时库的优化

runtime: use hardware divider to improve performance

```
The hardware divider is an optional component of ARMv7. This patch
detects whether it is available in runtime and use it or not.

1. The hardware divider is detected at startup and a flag is set/clear
   according to a perticular bit of runtime.hwcap.
2. Each call of runtime.udiv will check this flag and decide if
   use the hardware division instruction.

A rough test shows the performance improves 40-50% for ARMv7. And
the compatibility of ARMv5/v6 is not broken.
```

使用硬件除法器替代软件除法算法，除法性能提升40%；

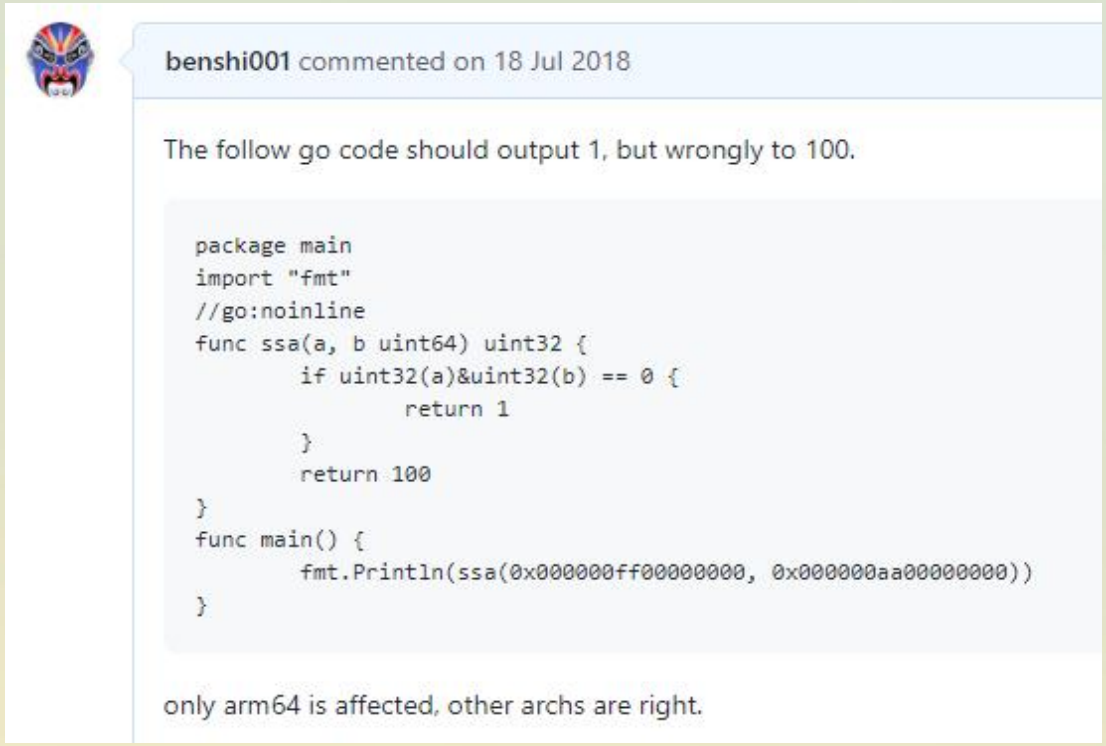
保持兼容性，ARMv5和ARMv6上使用软件除法，ARMv7上使用硬件除法；

程序启动时，动态探测当前硬件版本并选择除法实现；

https://github.com/golang/go/commits?author=benshi001

我发现及修复的bug

TST Ra, Rb； 64位位测试指令



benshi001 commented on 18 Jul 2018

The follow go code should output 1, but wrongly to 100.

```
package main
import "fmt"
//go:noinline
func ssa(a, b uint64) uint32 {
        if uint32(a)&uint32(b) == 0 {
                return 1
        }
        return 100
}
func main() {
        fmt.Println(ssa(0x00000ff00000000, 0x000000aa00000000))
}
```

only arm64 is affected, other archs are right.

TSTW Ra, Rb； 32位位测试指令

cmd/compile: fix an arm64's comparison bug

The arm64 backend generates "TST" for "if uint32(a)&uint32(b) == 0", which should be "TSTW".

fixes #26438

Change-Id: I7d64c30e3a840b43486bcd10eea2e3e75aaa4857
Reviewed-on: https://go-review.googlesource.com/124637
Run-TryBot: Ben Shi <powerman1st@163.com>
TryBot-Result: Gobot Gobot <gobot@golang.org>
Reviewed-by: Cherry Zhang <cherryyz@google.com>

master go1.15.4 ... go1.11beta2

benshi001 authored and cherrymui committed on 18 Jul 2018

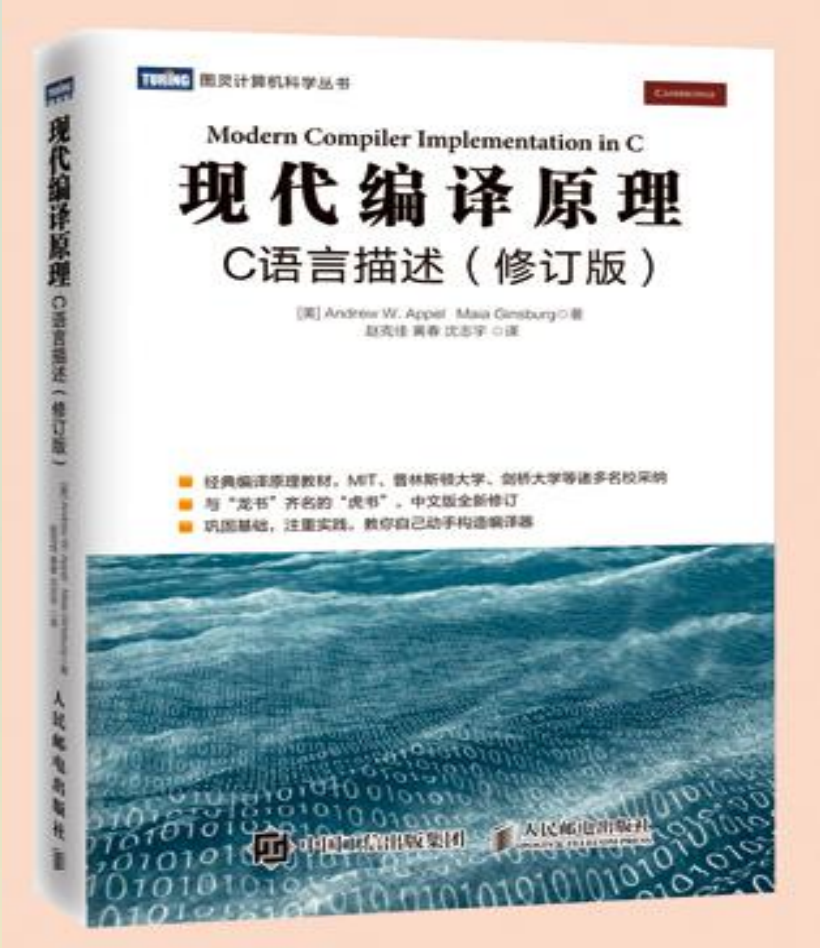
https://github.com/golang/go/issues/created_by/benshi001

学习编译器的益处

- 理解Go程序运行机制，提升代码质量
- 提升内功：词法分析涉及到正则表达式
- 提升内功：语法语义分析涉及到树变换和树搜索
- 提升内功：中端有大量的关于图变换和图搜索的高效算法
- 提升内功：掌握不同处理器/平台的特点
- 定制自己的DSL（Domain Specific Language）

编译器入门书籍


计 算 机 科 学 丛 书
可变目标C编译器
设计与实现
A Retargetable C Compiler
Design and Implementation


图灵计算机科学丛书
Modern Compiler Implementation in C
现代编译原理
C语言描述（修订版）
经典编译原理教材，MIT、普林斯顿大学、剑桥大学等诸多名校采纳
与“龙书”齐名的“虎书”，中文版全新修订
巩固基础，注重实践，教你自己动手构造编译器
人民邮电出版社

成为贡献者

- 选定一个自己擅长或有强烈兴趣的方向/领域/模块
- 确定该部分代码的owner，与其多交流

- 从修复issue开始：https://github.com/golang/go/issues/
- 从TODO开始：cd go/src/ & grep "TODO" * -ri
- 从添加测试用例开始
- 与其它语言/工具做对比
- 阅读代码，主动发现问题

成为贡献者

https://golangcn.org/

Golang China Contributor Club

If you have more than about 10 CLs are merged in Go repo, you can send an email to apply for membership.

You can also get an email account with the suffix @golangcn.org forever. Read more...

Members:

Baokun Lee (bk@golangcn.org) joined in 2019

Ben Shi (benshi@golangcn.org) joined in 2019

Changkun Ou (changkun@golangcn.org) joined in 2020

Cholerae Hu (cholerae@golangcn.org) joined in 2020

Fannie Zhang (fannie.zhang@golangcn.org) joined in 2020

Meng Zhuo (mzh@golangcn.org) joined in 2019

Shushan Chai (chai2010@golangcn.org) joined in 2020

Xiangdong Ji (xdji@golangcn.org) joined in 2020

Go官方对我的认可

Hope you can join us - confirm participation via this form.

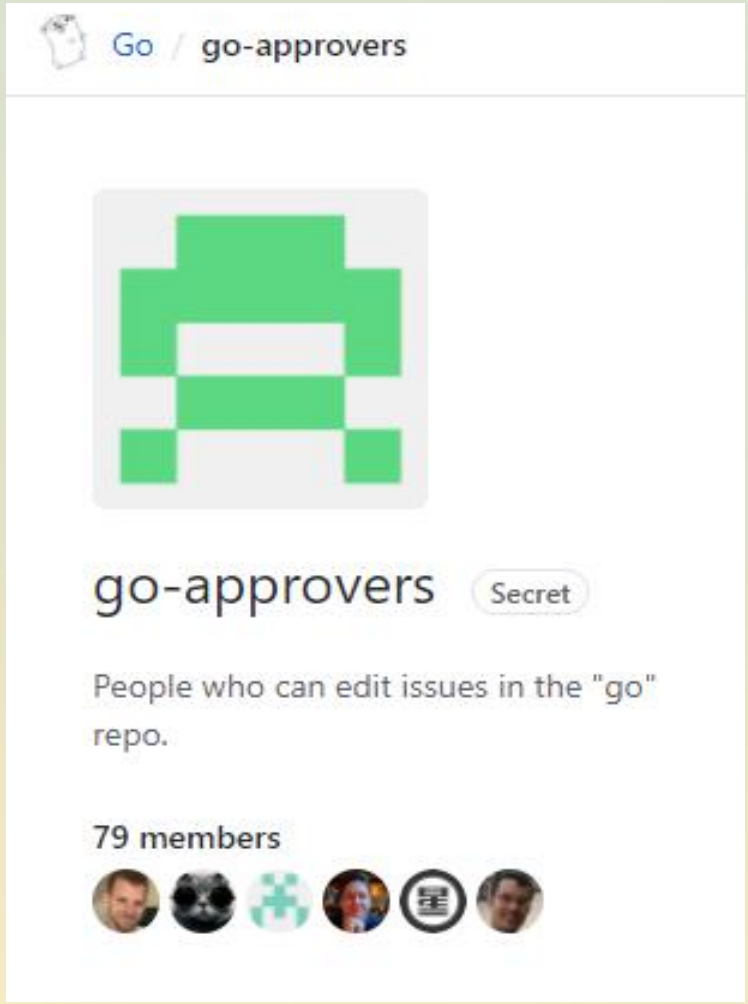
Go Contributor Summit Registration

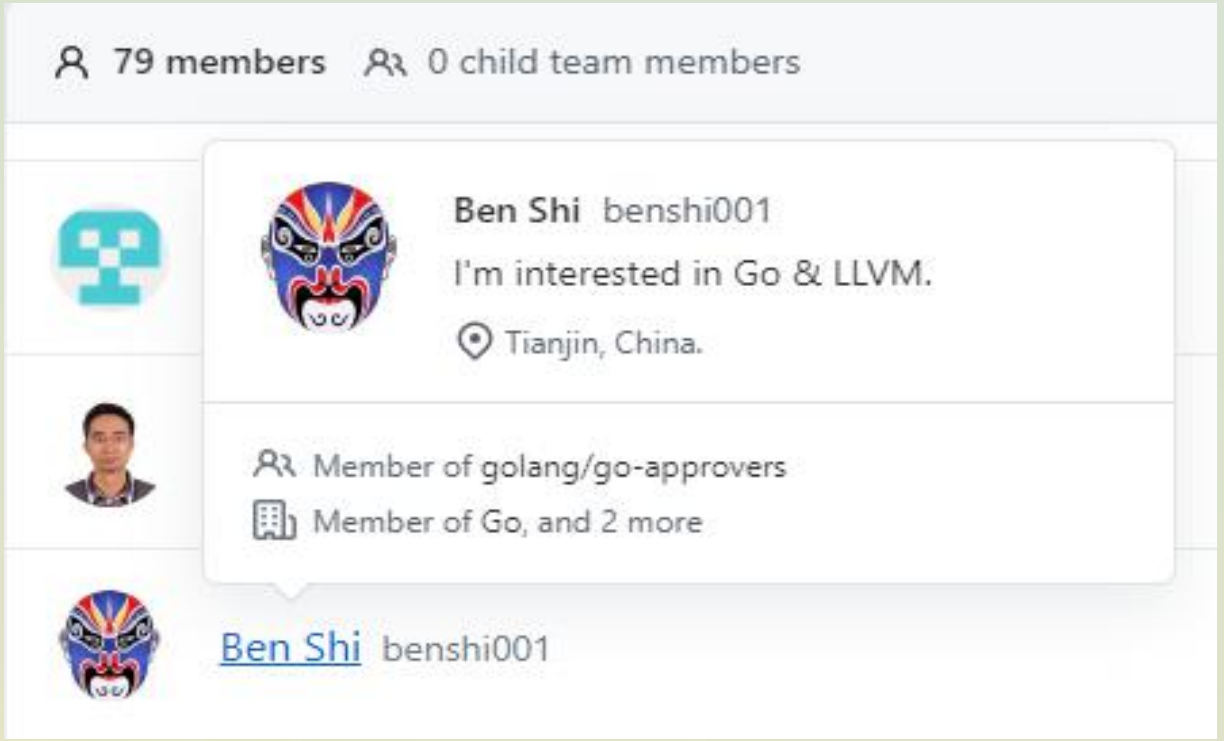
As a valued member of the Go community, we invite you to attend the Contributor Summit hosted at GopherCon San Diego. It will take place on Wednesday July 24th from 9a - 4p at the Mariott Marquis San Diego Marina. This event *does not* require a GopherCon ticket and is separate from other GopherCon activities.

The contributor summit is an opportunity for members of the community to gather together, discuss issues, and get face time with some of the Go core team.

被go core team邀请赴美参会

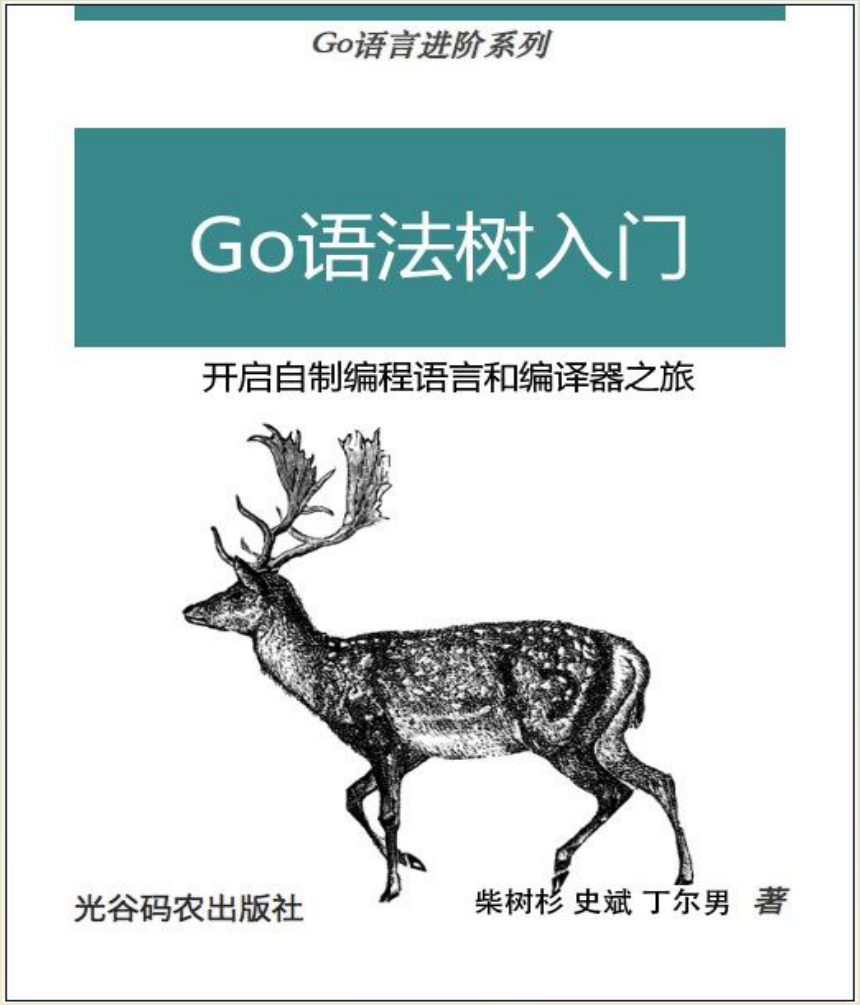
Go官方对我的认可





拥有go官方git仓库提交权限

Go官方对我的认可



Ian Lance Taylor 大佬真迹墨宝

Recommendation

The official "go/*" packages are important components of the Go programming language's tools for analyzing Go programs. They are a core part of programs like gofmt and go vet. Understanding these packages not only improves a gopher's programming skills, but can lead to building embedded scripts based on these packages.

Both Shushan Chai (chai2010@github) and Ben Shi (benshi001@github) are Go contributors, who have made many good commits to Go's master branch.

This book authored by them introduces the functionalities and also analyzes the implementation of the "go/*" packages.

I recommend that Chinese gophers read it and benefit from the content. What's more, I hope more Chinese gophers will make contributions to Go after reading it.


GOPHER CHINA 2020
中国 上海 / 2020-11.21-22
感谢聆听！