Почему Golang такой странный

Филипп Кулин (Дремучий лес)

Golang Conf 2019

Профессиональная конференция для Go-разработчиков

Откиньтесь на спинку кресла

- Эта презентация сделана с помощью LaTeX
- Я расскажу красивую лёгкую сказку
- Но в ней будет много несложного кода
- Весь код проверялся на компиляцию и работу

Go действительно странный

- Вот эти странные ”:=”и ”=”
- Нет дженериков
- Нет классического ООП, изобретено что-то свое
- Каналы какие-то
- Необычный язык ассемблера

Go действительно странный

- Вот эти странные ”:=”и ”=”
- Нет дженериков
- Нет классического ООП, изобретено что-то свое
- Каналы какие-то
- Необычный язык ассемблера

Зачем всё это было изобретать в 2007 году?

Рождённый старым

История Go началась задолго до 2007 года

В самом начале

- 1970-1975. Томпсон и Ритчи создают UNIX и C
- В 80-е бурно развиваются компьютерные сети
- К концу 80-х некоторые считают, что UNIX морально устарел

В самом начале

- 1970-1975. Томпсон и Ритчи создают UNIX и C
- В 80-е бурно развиваются компьютерные сети
- К концу 80-х некоторые считают, что UNIX морально устарел

“Not only is Unix dead, it’s starting to smell really bad”

В самом начале

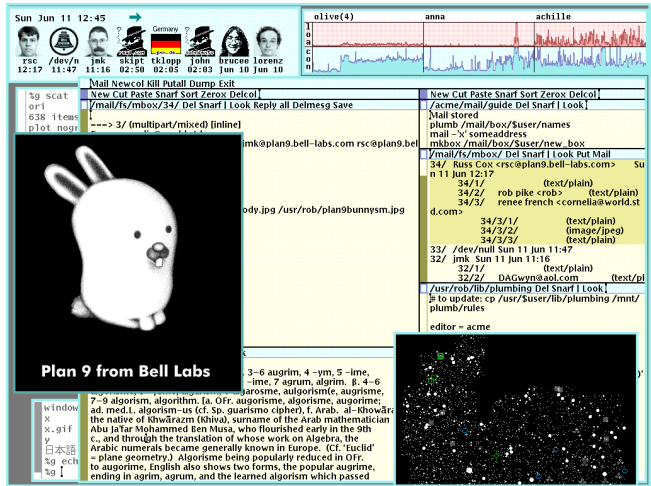
- 1970-1975. Томпсон и Ритчи создают UNIX и C
- В 80-е бурно развиваются компьютерные сети
- К концу 80-х некоторые считают, что UNIX морально устарел

“Not only is Unix dead, it’s starting to smell really bad”

— Роб Пайк, около 1991

Plan9 from Bell Labs

- 1989г — прототип
- 1992г — релиз
- Создан UTF-8
- Всё — файл, procfs
- Протокол 9P
- Make mainframe great again
- Несколько платформ



Plan9. Кросскомпиляция

- Компиляторы: 2c, 8c, 8al, 8g, 6a, 6c...
- Компоновщики: 2l, 8l, 6l...
- Первый символ обозначает архитектуру ”O”
- Компилятор создает объектный файл `file.<O>`
- Объектный файл — псевдо-ассемблер
- Компоновщик присваивает инструкции
- `% 6c -o test.6 test.c; 6l -o 6.test test.6`
- Или вот так: `% objtype=amd64 mk`

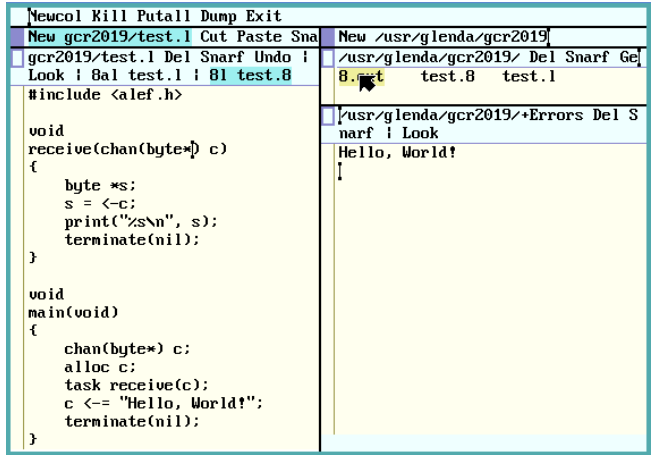
Архитектуры Plan9 (современные)

- **k**: SPARC
- **x**: AT&T DSP3210
- **v**: be MIPS 3000
- **q**: PowerPC
- **0**: le MIPS 3000

- **1**: Motorola 68000
- **2**: Motorola 68040
- **5**: le ARM
- **6**: AMD64
- **7**: ARM64
- **8**: i386

Редактор Acme

- Для разработчиков
- Автор — *Роб Пайк*
- Написан на Alef
- Alef = Acme
- 30 лет с нами



За несколько лет до этого...

- 1978 — *Тони Хоар*. Взаимодействие последовательных процессов
- 1982+ — *Роб Пайк*. Blit, mps, mux
- 1985 — *Лука Карделли* и *Роб Пайк*. Язык Squeak
- 1988 — *Роб Пайк*. Язык Newsqueak. Игрушечные окошки
- 1989-2002 — *Роб Пайк*. Окна 8½, rio
- 1989-1995 — *Фил Винтерботтом*. Язык Alef на основе Newsqueak

Язык Alef

- Нет сборщика мусора
- Частично написан на самом себе (рантайм)
- Компилятор создаёт стандартные объекты
- Параллельное и конкурентное программирование
- Последняя версия — 1995 год

Alef. Каналы

```
#include        <alef.h>

void receive(chan(byte*) c) {
        byte *s;
        s = <-c;
        print("%s\n", s);
        terminate(nil);
}

void main(void) {
        chan(byte*) c;
        alloc c;
        proc receive(c);
        c <-= "hello world";
        terminate(nil);
}
```

- Синхронные
- Асинхронные
- Всё, как мы любим
- И между процессами
- И между корутинами

Alef. Параллелизм

```
chan(int) a0, a1;

void produce(chan(int) j, int c);

void main(void) {
    int i;
    alloc a0, a1;
    proc produce(a0, '0');
    proc produce(a1, '1');
    for(i = 0; i < 10; i++)
        par {
            a0 <-= i;
            a1 <-= i;
        }
}
```

`proc`: запуск процесса
`par`: параллельный запуск

Alef. Хороший пример ”par”

```
active = malloc(BUFSIZE);
passive = malloc(BUFSIZE);

n = decode(fd, active, BUFSIZE);     /* first block */
while(active != nil) {
    par {
        display(active, n);

        if(decode(fd, passive, BUFSIZE) <= 0)
            passive = nil;
    }
    (active, passive) = (passive, active);
}
```

Alef. Конкурентность

```
void produce(chan(int) c);

void main(void) {
  int a,i;
  chan (int) c1, c2;
  alloc c1, c2;
  task produce(c1),produce(c2);
  for(i = 0; i < 10 ; i++)
    alt {
        case a =<-c1:
          print(”1 -> %d\n”,a);
          break;
        case a =<-c2:
          print(”2 -> %d\n”,a);
          break;
    };
}
```

Alef. Планировщик

- Процессы (`proc`):
  - планируются системой
  - синхронизируются rendezvous(2)

Alef. Планировщик

- Процессы (`proc`):
  - планируются системой
  - синхронизируются rendezvous(2)
- Корутины (`task`):
  - переключаются: `QLock.lock, Rendez.sleep, alt, <-=, <-`

Alef. Планировщик

- Процессы (`proc`):
  - планируются системой
  - синхронизируются rendezvous(2)
- Корутины (`task`):
  - переключаются: `QLock.lock, Rendez.sleep, alt, <-=, <-`
- Коммуникация через каналы

Alef. Возврат функций

```
int i,j;
float f;

tuple(int, int, float) test(int c) {
        return (c, c+1, (float)c/(float)(c+1));
}

(i, j, f) = test(1);
```

Alef. Обработка ошибок

```
alloc p;

rescue {
        unalloc p;
        return nil;
}

...

if(error)
        raise;
return p;
```

- Блок в контексте функции
- Явный вызов
- Закрытие файлов, unalloc

Alef. Полиморфный тип данных

Равнозначная структура:

```
aggr Polytype
{
        void*   ptr;
        int     tag;
};
```

Особенности:

```
typedef Interface;

Interface a, b, c, d;
int i;

a = (alloc Interface)10;
b = (alloc Interface)"Hello";
d = (alloc Interface)i;
c = a;  /* copy pointer */
d := b; /* copy value */
```

Alef. Полиморфный тип данных

Равнозначная структура:

```
aggr Polytype
{
        void*   ptr;
        int     tag;
};
```

Особенности:

```
typedef Interface;

Interface a, b, c, d;
int i;

a = (alloc Interface)10;
b = (alloc Interface)"Hello";
d = (alloc Interface)i;
c = a;  /* copy pointer */
d := b; /* copy value */
```

Alef. Полиморфный тип данных

Равнозначная структура:

```
aggr Polytype
{
        void*   ptr;
        int     tag;
};
```

Особенности:

```
typedef Interface;

Interface a, b, c, d;
int i;

a = (alloc Interface)10;
b = (alloc Interface)"Hello";
d = (alloc Interface)i;
c = a;  /* copy pointer */
d := b; /* copy value */
```

Alef. Полиморфный тип данных

Равнозначная структура:

```
aggr Polytype
{
        void*   ptr;
        int     tag;
};
```

Особенности:

```
typedef Interface;

Interface a, b, c, d;
int i;

a = (alloc Interface)10;
b = (alloc Interface)"Hello";
d = (alloc Interface)i;
c = a;  /* copy pointer */
d := b; /* copy value */
```

Alef. Полиморфный тип данных

Равнозначная структура:

```
aggr Polytype
{
        void*   ptr;
        int     tag;
};
```

Особенности:

```
typedef Interface;

Interface a, b, c, d;
int i;

a = (alloc Interface)10;
b = (alloc Interface)"Hello";
d = (alloc Interface)i;
c = a;  /* copy pointer */
d := b; /* copy value */
```

Alef. Программирование

- На Alef не очень хорошая документация
- Руководство неплохое, но не полное
- Чтобы начать писать, нужно буквально два вечера

Alef. Необычный ООП

```
adt Point {
    int x,y;
    void set(*Point,int,int);
};
adt Circle {
            Point;
    extern int radius;
            void set(*Circle,int,int,int);
    intern int tst(*Circle);
};
...
void
Circle.set(Circle *c, int x, int y, int r)
{
    c->Point.set(x,y);
    c->radius = r;
}
```

```
...
Circle c;
c.set(5,4,3);
```

- **Абстрактный тип**

Alef. Необычный ООП

```
adt Point {
    int x,y;
    void set(*Point,int,int);
};
adt Circle {
            Point;
    extern int radius;
            void set(*Circle,int,int,int);
    intern int tst(*Circle);
};
...
void
Circle.set(Circle *c, int x, int y, int r)
{
    c->Point.set(x,y);
    c->radius = r;
}
```

```
...
Circle c;
c.set(5,4,3);
```

- Абстрактный тип
- **Поля**

Alef. Необычный ООП

```
adt Point {
    int x,y;
    void set(*Point,int,int);
};
adt Circle {
            Point;
    extern int radius;
            void set(*Circle,int,int,int);
    intern int tst(*Circle);
};
...
void
Circle.set(Circle *c, int x, int y, int r)
{
    c->Point.set(x,y);
    c->radius = r;
}
```

```
...
Circle c;
c.set(5,4,3);
```

- Абстрактный тип
- Поля
- **Методы**

Alef. Необычный ООП

```
adt Point {
    int x,y;
    void set(*Point,int,int);
};
adt Circle {
            Point;
    extern int radius;
           void set(*Circle,int,int,int);
    intern int tst(*Circle);
};
...
void
Circle.set(Circle *c, int x, int y, int r)
{
    c->Point.set(x,y);
    c->radius = r;
}
```

```
...
Circle c;
c.set(5,4,3);
```

- Абстрактный тип
- Поля
- Методы
- **Встраивание**

Alef. Необычный ООП

```
adt Point {
    int x,y;
    void set(*Point,int,int);
};
adt Circle {
            Point;
    extern int radius;
            void set(*Circle,int,int,int);
    intern int tst(*Circle);
};
...
void
Circle.set(Circle *c, int x, int y, int r)
{
    c->Point.set(x,y);
    c->radius = r;
}
```

```
...
Circle c;
c.set(5,4,3);
```

- Абстрактный тип
- Поля
- Методы
- Встраивание
- **Скрытие**

Alef. Необычный ООП

```
adt Point {
    int x,y;
    void set(*Point,int,int);
};
adt Circle {
            Point;
    extern int radius;
            void set(*Circle,int,int,int);
    intern int tst(*Circle);
};
...
void
Circle.set(Circle *c, int x, int y, int r)
{
    c->Point.set(x,y);
    c->radius = r;
}
```

```
...
Circle c;
c.set(5,4,3);
```

- Абстрактный тип
- Поля
- Методы
- Встраивание
- Скрытие
- **Передача ссылки**

Alef. Дженерики

- **Абстрактный тип с параметром**

```
adt Stack[T] {
    extern int  tos;
           T    data[100];
           void push(*Stack, T);
};

Stack[int] bound;
Stack unbound;

/* type casting */
bound.f(3);
unbound.f((alloc T)3);
```

```
int i, j, k;

i := bound.data[bound.tos];
j = bound.data[bound.tos];
k = bound.data[bound.tos]; /* err */
```

Alef. Дженерики

- Абстрактный тип с параметром
- **Полиморфный тип и приведение типов**

```
adt Stack[T] {
    extern int  tos;
           T    data[100];
           void push(*Stack, T);
};

Stack[int] bound;
Stack unbound;

/* type casting */
bound.f(3);
unbound.f((alloc T)3);
```

```
int i, j, k;

i := bound.data[bound.tos];
j = bound.data[bound.tos];
k = bound.data[bound.tos]; /* err */
```

Alef. Дженерики

- Абстрактный тип с параметром
- **Полиморфный тип и приведение типов**

```
adt Stack[T] {
    extern int  tos;
           T    data[100];
           void push(*Stack, T);
};

Stack[int] bound;
Stack unbound;

/* type casting */
bound.f(3);
unbound.f((alloc T)3);
```

```
int i, j, k;

i := bound.data[bound.tos];
j = bound.data[bound.tos];
k = bound.data[bound.tos]; /* err */
```

Alef. Дженерики

- Абстрактный тип с параметром
- Полиморфный тип и приведение типов

```
adt Stack[T] {
    extern int  tos;
           T    data[100];
           void push(*Stack, T);
};

Stack[int] bound;
Stack unbound;

/* type casting */
bound.f(3);
unbound.f((alloc T)3);
```

```
int i, j, k;

i := bound.data[bound.tos];
j = bound.data[bound.tos];
k = bound.data[bound.tos]; /* err */
```

- ”:=”: без освобождения

Alef. Дженерики

- Абстрактный тип с параметром
- Полиморфный тип и приведение типов

```
adt Stack[T] {
    extern int  tos;
           T    data[100];
           void push(*Stack, T);
};

Stack[int] bound;
Stack unbound;

/* type casting */
bound.f(3);
unbound.f((alloc T)3);
```

```
int i, j, k;

i := bound.data[bound.tos];
j = bound.data[bound.tos];
k = bound.data[bound.tos]; /* err */
```

- ":=": без освобождения
- **"=": с освобождением**

Alef. Biobuf

```
#include <alef.h>
#include <bio.h>

Biobuf stdout;

void
main(void)
{
        stdout.init(1, OWRITE);
        stdout.print(”hello world!\n”);
}
```

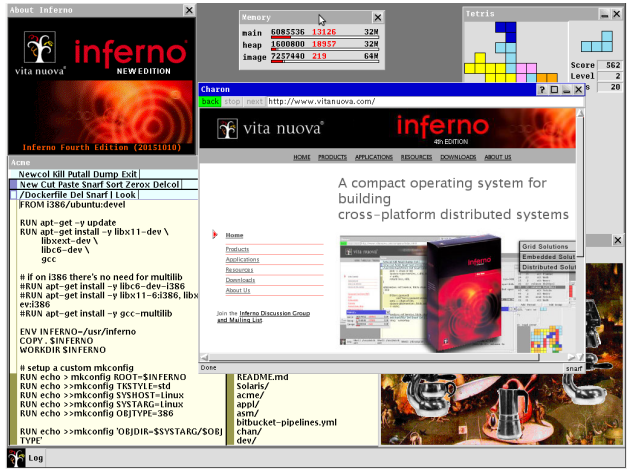
Заброшен и забыт

- 1995г — Plan9 2-nd edition
- 1995г — Windows'95
- 1995г — Java
- Bell Labs бросается создавать конкурента Java
- Plan9 заброшен
- Alef прекращает существование
  - Нет сборщика мусора
  - Трудно поддерживать
  - По слухам, дженерики трудно реализовать со сборщиком мусора

Inferno

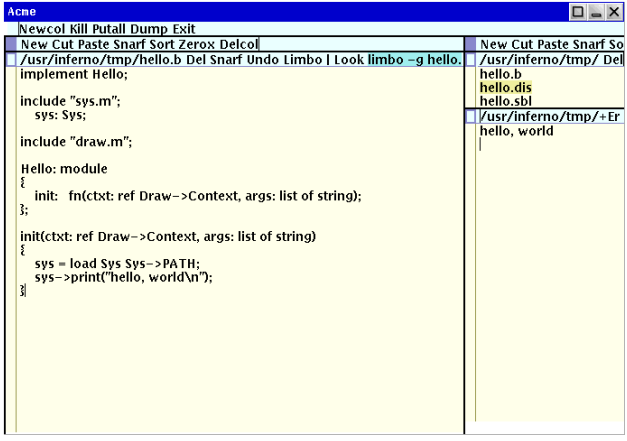
Земную жизнь пройдя до половины,
Я очутился в сумрачном лесу,
Утратив правый путь во тьме долины.

- 1996 — версия 1.0
- Наследник Plan9
- Новый 9P — Styx
- Регистровая Dis VM



Редактор Acme

- Для разработчиков
- Автор — *Роб Пайк*
- Переписан на Limbo
- 30 лет с нами



Limbo

- ”JIT-ed Newsqueak”*(Роб Пайк, 2007)*
- Сборщик мусора, реализованный в виртуальной машине
- Планировщик — виртуальная машина
- Результат — байткод для виртуальной машины
- Только вытесняющие процессы
- Идеология модулей
- В последнем релизе появились дженерики

Limbo. Некоторые замечания

- Крайне мало документации
- Некоторый функционал ограничен
- Использовался в промышленных продуктах
- Имеет много сохранившегося кода
- Работать с кодом Limbo непривычно и сложно
- Go — прямой потомок Limbo? Посмотрим

Limbo. Hello, World!

```
implement Hello;

include "sys.m";
        sys: Sys;
include "draw.m";

Hello: module
{
        init:   fn(ctxt: ref Draw->Context, argv: list of string);
};

init(ctxt: ref Draw->Context, argv: list of string)
{
        sys = load Sys Sys->PATH;
        sys->print("Hello World\n");
}
```

Limbo. Примеры синтаксиса

```
i, j: int;
k := 1;          # int
b := byte 2;     # byte

s: string;       # длина в символах
s1 := s[0:];     # срез

a: array of int;
a = array[10] of int;

myfunc: fn(i, k: int, s: string) : (list of string, int);
```

Limbo. Примеры синтаксиса

```
i, j: int;
k := 1;          # int
b := byte 2;     # byte

s: string;       # длина в символах
s1 := s[0:];     # срез

a: array of int;
a = array[10] of int;

myfunc: fn(i, k: int, s: string) : (list of string, int);
```

Limbo. Примеры синтаксиса

```
i, j: int;
k := 1;          # int
b := byte 2;     # byte

s: string;       # длина в символах
s1 := s[0:];     # срез

a: array of int;
a = array[10] of int;

myfunc: fn(i, k: int, s: string) : (list of string, int);
```

Limbo. Примеры синтаксиса

```
i, j: int;
k := 1;          # int
b := byte 2;     # byte

s: string;       # длина в символах
s1 := s[0:];     # срез

a: array of int;
a = array[10] of int;

myfunc: fn(i, k: int, s: string) : (list of string, int);
```

Limbo. Примеры синтаксиса

```
i, j: int;
k := 1;          # int
b := byte 2;     # byte

s: string;       # длина в символах
s1 := s[0:];     # срез

a: array of int;
a = array[10] of int;

myfunc: fn(i, k: int, s: string) : (list of string, int);
```

Limbo. Дженерики

```
reverse[T](l: list of T): list of T
{
        rl: list of T;
        for(; l != nil; l = tl l)
                rl = hd l :: rl;
        return rl;
}

Tree: adt[T] {
    v: T;
    l, r: cyclic ref Tree[T];
};
```

Limbo. Bufio

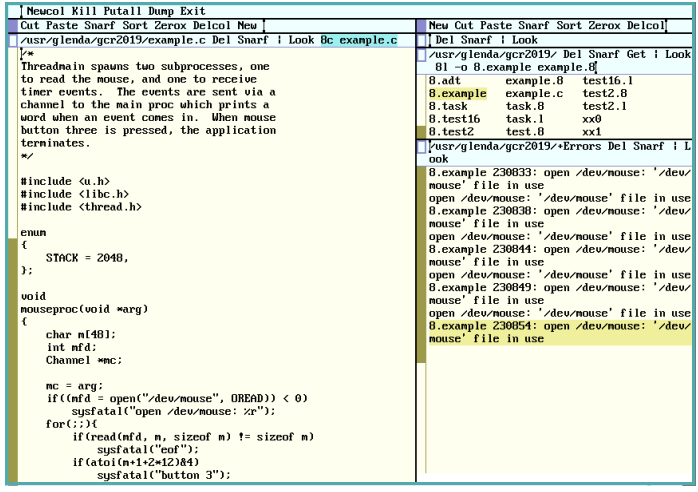
```
init(nil: ref Draw->Context, nil: list of string)
{
    sys := load Sys Sys->PATH;
    bufio = load Bufio Bufio->PATH;

    bout = bufio->fopen(sys->fildes(1), Bufio->OWRITE);
    bout.puts("Hello World\n");

    bout.flush();
}
```

Резкий разворот

- Inferno не «взлетел» и продан Vita Nuova
- Lucent делает новый релиз Plan9 в 2000 году
- Библиотека libthread на замену Alef

Редактор Acme

- Для разработчиков
- Автор — *Роб Пайк*
- C и libthread
- 30 лет с нами



libthread

- Написана на C
- Работает и с процессами, и с корутинами
- Корутины переключаются: `yield, proccreate, procexec, procexecl, threadexits, alt, send, recv (и основанными на send и recv), qlock, rlock, wlock, rsleep`
- Коммуникация через каналы

libbio. Подсчет рун

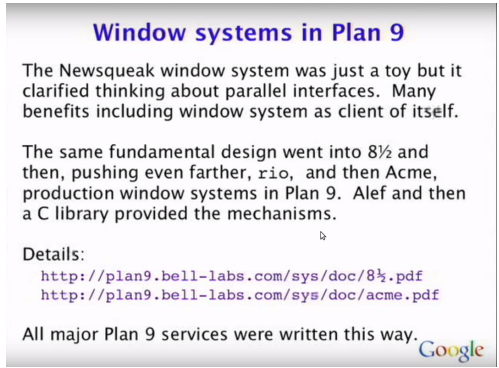
```
uvlong runecount(Biobuf *f, char *filename) {
    uvlong n;
    Rune r;
    n = 0;

    while((r = Bgetrune(f)) != (Rune)Beof)
        n++;

    print("%10ulld %s\n", n, filename);

    return n;
}
```

Наши дни

Вселенная Plan9

Конкурентное программирование на примере Newsqueak

- Март 2007 года. Google
- Newsqueak из 1988
- В сентябре 2007...



https://youtu.be/hB05UFqOtFA

Редактор Acme

- Для разработчиков
- Автор — ???
- Переписать на Go
- 30 лет с нами

Go

- Сентябрь 2007 года — начало разработки
- Первые версии собирались только на Plan9
- Из Go, пока он был на C, «торчали уши» Plan9
- Отказался от процессов и дженериков
- Взял ООП, каналы, корутины и кросскомпиляцию
- Авторы Go всё знают и хорошо подумали
- Если что-то не ясно в Go — поищите в его истории
- Отличный пример — Go-ассемблер

Использования Go-ассемблера

Только если вы **отлично** представляете, что делаете

- Cloudflare CIRCL
  - blog.cloudflare.com/introducing-circl/
- Шифр «Кузнечик» на языке Go
  - dxdt.ru/2019/02/18/8702/
  - Прямое использование AVX

Plan9. Кросскомпиляция

- Компиляторы: 2c, 8c, 8al, 8g, 6a, 6c...
- Компоновщики: 2l, 8l, 6l...
- Первый символ обозначает архитектуру ”O”
- Компилятор создает объектный файл `file.<O>`
- Объектный файл — псевдо-ассемблер
- Компоновщик присваивает инструкции
- `% 6c -o test.6 test.c; 6l -o 6.test test.6`
- Или вот так: `% objtype=amd64 mk`

Язык ассемблера Go

- Официальный сайт отсылает к Plan9
- Это все же не совсем Plan9-ассемблер
- Псевдо-ассемблер
- Пайк про Go-ассемблер: youtu.be/KINIAgRpkDA
- Полезные таблицы: quasilyte.dev/blog/post/go-asm-complementary-reference/
- Понимание принципов компиляции в Plan9 помогло

Вопросы

Авторы Go всё знают и хорошо подумали
Если что-то не ясно в Go — поищите в его истории

schors@gmail.com

Ссылки. Plan9

[1] *Plan9 from Bell Labs*. http://9p.io/.

[2] Doug McIlroy. *Plan9. Preface to the Second (1995) Edition*. Март 1995. http://9p.io/sys/man/preface.html.

[3] *Standalone term/cpu/auth/file servers*. http://www.plan9.fi/cpu.txt.

[4] *A fork of Plan 9 from Bell Labs by the People's Front of Cat -V*. http://9front.org/.

[5] *building Alef language*. http://mail.9fans.net/pipermail/9fans/2019-September/037921.html.

[6] *Plan9 THREAD(2)*. http://man.cat-v.org/plan_9/2/thread.

Ссылки. Inferno

[7] *Inferno OS*. http://www.vitanuova.com/inferno/.

[8] *Very Concurrent Mark and Sweep Garbage Collection without Fine-Grain Synchronization*. http://doc.cat-v.org/inferno/concurrent_gc/.

[9] *Addendum to The Limbo Programming Language*. http://www.vitanuova.com/inferno/papers/addendum.pdf.

[10] *System and Interface Changes to Inferno*. http://doc.cat-v.org/inferno/4th_edition/release_notes/changes.

[11] *OS Inferno: Programming Limbo (RU)*. https://powerman.name/Inferno/Limbo.html.

[12] *Examples for the Limbo Programming Language*. https://github.com/henesy/limbobyexample.

Ссылки. Разное

[13] Tony Hoare. *Communicating Sequential Processes (CSP)*. 18 мая 1978. http://www.usingcsp.com/.

[14] Rob Pike. *Concurrency/message passing Newsqueak*. Март 2007. https://youtu.be/hB05UFqOtFA.

[15] *go/go/refs/tags/weekly.2009-11-06*. 6 нояб. 2009. https://go.googlesource.com/go/+/refs/tags/weekly.2009-11-06.

[16] *Bell Labs and CSP Threads*. https://swtch.com/~rsc/thread/.

[17] *Interview with Dennis Ritchie (2003)*. Февр. 2003. https://anders.unix.se/2015/10/26/interview-with-dennis-ritchie-2003/.

[18] Тристан Хьюм. *Модели дженериков и метапрограммирования: Go, Rust, Swift, D и другие*. 25 июля 2019. https://habr.com/ru/company/mailru/blog/461321/.

[19] Ян Ланс Тейлор. *Зачем нужны дженерики в Go?*. 7 авг. 2019. https://habr.com/ru/company/mailru/blog/462811/.

Ссылки. Ассемблер

[20] *A Quick Guide to Go's Assembler*. https://golang.org/doc/asm.

[21] *A Manual for the Plan 9 assembler*. https://9p.io/sys/doc/asm.html.

[22] Rob Pike. *The Design of the Go Assembler*. https://youtu.be/KINIAgRpkDA.

[23] Александр Венедюхин. *Шифр «Кузнечик» на Go*. 18 февр. 2019. https://dxdt.ru/2019/02/18/8702/.

[24] *Cloudflare CIRCL*. 20 июня 2019. https://blog.cloudflare.com/introducing-circl/.

[25] Iskander Sharipov. *Go assembly language complementary reference*. 20 сент. 2017. https://quasilyte.dev/blog/post/go-asm-complementary-reference/.