Lygiagretusis programavimas Google Go kalba

Go kalbos paskirtis ir ypatybės

- sistemų projektavimo kalba, bet gali būti naudojama kaip bendrosios paskirties kalba;
- atvirojo kodo projektas;
- nėra klasių, tačiau kiekvienam tipui galima kurti metodus (funkcijas);
- palaiko lygiagretumą (*concurrency*);
- OS: Unix, Windows ir kt.
- Galima sakyti, kad Google Go – tai XXI amžiaus C kalba.

Autoriai

- **Robert Griesemer**
  Darbovietės: Google Inc.
  Projektai: Limbo, Go programavimo kalbos, Java HotSpot virtuali mašina;
- **Rob Pike**
  Darbovietės: Bell Labs, Google Inc.
  Projektai: Limbo, Go programavimo kalbos, Unix OS;
- **Ken Thompson**
  Darbovietės: Bell Labs, Entrisphere Inc., Google Inc.
  Projektai: B, C, Go programavimo kalbos, Unix OS.

- **www**: http://golang.org/.

Istorija

- **2007** m. rugsėjis – Go projekto pradžia;
- **2009** m. lapkritis – oficialus Go pristatymas;
- **2012** m. kovas – 1.0 versija;
- **2013** m. gegužė – 1.1 versija;
- **2013** m. gruodis – 1.2 versija;
- **2014** m. birželis – 1.3 versija;
- **2014** m. gruodis – 1.4 versija;
- **2015** m. rugpjūtis – 1.5 versija;
- **2016** m. vasaris – 1.6 versija;
- **2016** m. rugpjūtis – 1.7 versija;

- **2009** m. - "Metų programavimo kalba" titulas (TIOBE Programming Community Index) už sparčiausią reitingo kilimą (http://www.tiobe.com/tiobe-index).

Pirmoji Go programa

```
package main
import fmt "fmt"

func main() {
  fmt.Printf("Hello, world")
}
```

Programavimo priemonės

- `go` – Go programų tvarkymo įrankis:
  - `go build ...` – kompiliuoti programą (paketą),
  - `go run ...` – kompiliuoti ir vykdyti programą (paketą);
  - `go ...` – `....`
- **GoClipse** – Eclise programų kūrimo aplinkos įskiepis.
- **Go for NetBeans** – NetBeans programų kūrimo aplinkos įskiepis.

Go & C++: sąvokų skirtumai

- Go kalboje nėra klasių.
- Yra rodyklės, bet nėra rodyklių aritmetikos.
- Jei masyvas yra funkcijos parametras, kreipinys į funkciją daro masyvo kopiją.
- Eilutės (`string`) ir atvaizdžiai (`map`) yra kalbos dalis. Po sukūrimo eilutės negali būti keičiamos.
- Nėra antraščių (`header`) failų, tačiau yra paketai (`package`).
- Operacijų operandais negali būti skirtingų tipų reikšmės (reikia naudoti tipų konversiją).
- Negalima užkloti funkcijų ir kurti savo operatorius.
- Nėra `while` ciklų.
- Vietoje `NULL` naudojamas `nil`.
- Lygiagretumas yra kalbos dalis nuo 1-ojo kalbos varianto.

Go & C++: sintaksės skirtumai. Programos tekstas

- Tekste be apribojimų galima rašyti Unikodo simbolius.
- Programos eilutės pabaigoje ir prieš uždarantį skliaustą galima nerašyti kabliataškio.
- Atidarantis riestinis skliaustas `{` rašomas toje pačioje eilutėje, kaip ir atitinkamas žodis (`if`, `for`, `switch`, `select`).
- ir kt.

Go & C++: sintaksės skirtumai. Kintamųjų aprašai

C++	Go
`int v1, v2;`	`var v1, v2 int`
`int *v3, *v4;`	`var v3, v4 *int`
`int v5[10];`	`var v5 [10]int`
`struct {int f;} v6;`	`var v6 struct {f int}`

Go kalbos galimybės

C++

- ???
- ???

Go

```
x, y = y, x
```

```
func f() (i int, j int) { ... }
v1, v2 = f()
```

Funkcijos reikšmių grąžinimas

```
func f() (int, int) {
  return 1, 2
}
func main() {
  a, b := f()
}
```

Masyvo panaudojimas

```
func main() {
  a := [5]float64{ 98, 93, 77, 82, 83 }
  var suma float64 = 0
  for _, reikšmė := range a {
    suma += reikšmė
  }
  fmt.Println(suma / float64(len(a)))
}
```

Go kalbos paprogramės (*goroutines*)

- Go programos funkcijos gali būti kviečiamos ne tik įprastu būdu, bet ir naudojant go sakinį;
- go sakiniuose kviečiamos funkcijos vykdomos kaip atskiros gijos, t.y., lygiagrečiai (*concurrent*)

Lygiagretus go vykdymas

```
func main() {
  ftm.Println("Dirba 10 procesų.")
  for i := 0; i < 10; i++ {
    go Procesas(i)
  }
  fmt.Println("Programa baigė darbą.")
}
func Procesas(k int) {
  fmt.Printf("Procesas nr.:  %d\n", k)
}
```

Galimi rezultatai

```
Dirba 10 procesų.
Programa baigė darbą.
Procesas nr.:  1
Procesas nr.:  0
Procesas nr.:  3
Procesas nr.:  4
Procesas nr.:  6
Procesas nr.:  2
Procesas nr.:  5
Procesas nr.:  8
Procesas nr.:  9
Procesas nr.:  7
```

Komunikavimas tarp *goroutines*

- komunikavimui tarp *goroutines* taikomas Hoare CSP (occam, JavaCSP) modelis;
- komunikavimui naudojami tipizuoti kanalai:
  - nebuferizuotas: `ch := make(chan Tipas)`
    buferizuotas: `ch := make(chan Tipas, n)`
  - rašymas į kanalą (dvinarė operacija): `ch <- a`
  - skaitymas iš kanalo (vienanarė operacija): `<- ch`
- sinchronizavimui naudojami nebuferizuoti (sinchroniniai) kanalai.

Sinchroninio kanalo panaudojimo pavyzdys (1)

```
func main() {
  ch := make(chan int)
  baigti := make(chan int)
  go Siuntėjas(ch)
  go Gavėjas(ch, baigti)
  <-baigti
}
func Siuntėjas(kanalas chan int) {
  x := 10
  kanalas <- x
}
func Gavėjas(kanalas chan int, baigti chan int) {
  a := <- kanalas
  baigti <- 0
}
```

Sinchroninio kanalo panaudojimo pavyzdys (2)

```
read-write
romas@romas-VirtualBox:~/Go/00-Read-Write/bin$ ./read-write
Dirba 2 procesai: Siuntėjas ir Gavėjas.
Siunčiu  x = 10
Pradinis a = 99
Gautas   a = 10
Programa baigė darbą.
romas@romas-VirtualBox:~/Go/00-Read-Write/bin$ ./read-write
Dirba 2 procesai: Siuntėjas ir Gavėjas.
Siunčiu  x = 10
Pradinis a = 99
Gautas   a = 10
Programa baigė darbą.
```

Asinchrononis kanalas (1)

```
func main() {
  var ch chan int
  ch = make(chan int, 5)
  baigti := make(chan int, 0)
  go Gavėjas(ch, baigti)
  go Siuntėjas(ch, "1 siuntėjas", 11111)
  go Siuntėjas(ch, "2 siuntėjas", 2222)
  go Siuntėjas(ch, "3 siuntėjas", 333)
  go Siuntėjas(ch, "4 siuntėjas", 44)
  go Siuntėjas(ch, "5 siuntėjas", 5)
  <-baigti
}
```

Asinchrononis kanalas (2)

```
func Siuntėjas(kanalas chan int,
    vardas string, x int) {
  kanalas <- x
}
```

Asinchrononis kanalas (3)

```
func Gavėjas(kanalas chan int,
    baigti chan int) {
  a := 99
  for i := 0; i < 5; i++ {
    a = <- kanalas
  }
  baigti <- 0
}
```

Asinchrononis kanalas (4)

```
read-write-any
romas@romas-VirtualBox:~/Go/01-Read-Write-Any/bin$ ./read-write-any
Dirba 6 procesai:  5 Siuntėjai ir Gavėjas. Buferizuoti kanalai
---Pradinis a = XXXXXXX, 99
2 siuntėjas siunčiu:   2222
1 siuntėjas siunčiu:  11111
4 siuntėjas siunčiu:     22
5 siuntėjas siunčiu:      5
3 siuntėjas siunčiu:    333
**** 2 siuntėjas atsiuntė:   2222
**** 1 siuntėjas atsiuntė:  11111
**** 5 siuntėjas atsiuntė:      5
**** 4 siuntėjas atsiuntė:     22
**** 3 siuntėjas atsiuntė:    333
Programa baigė darbą.
romas@romas-VirtualBox:~/Go/01-Read-Write-Any/bin$ ./read-write-any
Dirba 6 procesai:  5 Siuntėjai ir Gavėjas. Buferizuoti kanalai
---Pradinis a = XXXXXXX, 99
1 siuntėjas siunčiu:  11111
2 siuntėjas siunčiu:   2222
3 siuntėjas siunčiu:    333
4 siuntėjas siunčiu:     22
5 siuntėjas siunčiu:      5
**** 1 siuntėjas atsiuntė:  11111
**** 2 siuntėjas atsiuntė:   2222
**** 3 siuntėjas atsiuntė:    333
**** 4 siuntėjas atsiuntė:     22
**** 5 siuntėjas atsiuntė:      5
Programa baigė darbą.
```

Sinchrononis kanalas. Alternatyva (1)

```
func main() {
  var ch [5] chan int
  for i := 0; i < 5; i++ {
    ch[i] = make(chan int, 0)
  }
  baigti := make(chan int, 0)
  go Gavėjas(& ch, baigti)
  go Siuntėjas(ch[0], "1 siuntėjas", 11111)
  go Siuntėjas(ch[1], "2 siuntėjas", 2222)
  go Siuntėjas(ch[2], "3 siuntėjas", 333)
  go Siuntėjas(ch[3], "4 siuntėjas", 44)
  go Siuntėjas(ch[4], "5 siuntėjas", 5)
  <-baigti
}
```

Sinchrononis kanalas. Alternatyva (2)

```
func Siuntėjas(kanalas chan int,
    vardas string, x int) {
  kanalas <- x
}
```

Sinchrononis kanalas. Alternatyva (3)

```
func Gavėjas(kanalas *[5]chan int,
    baigti chan int) {
  a := 99
  for i := 0; i < 5; i++ {
    select {
     case a = <- kanalas[0]:
     case a = <- kanalas[1]:
     case a = <- kanalas[2]:
     case a = <- kanalas[3]:
     case a = <- kanalas[4]:
    }
  }
  baigti <- 0
}
```

Sinchrononis kanalas. Alternatyva (4)

```
alternatyva
romas@romas-VirtualBox:~/Go/01-Alternatyva/bin$ ./alternatyva
Dirba 6 procesai:  5 Siuntėjai ir Gavėjas.
---Pradinis a = 99
2 siuntėjas: siunčiu:   2222
1 siuntėjas: siunčiu:  11111
3 siuntėjas: siunčiu:    333
4 siuntėjas: siunčiu:     44
5 siuntėjas: siunčiu:      5
**** 2 siuntėjas atsiuntė:   2222
**** 3 siuntėjas atsiuntė:    333
**** 4 siuntėjas atsiuntė:     44
**** 5 siuntėjas atsiuntė:      5
**** 1 siuntėjas atsiuntė:  11111
Programa baigė darbą.
romas@romas-VirtualBox:~/Go/01-Alternatyva/bin$ ./alternatyva
Dirba 6 procesai:  5 Siuntėjai ir Gavėjas.
---Pradinis a = 99
1 siuntėjas: siunčiu:  11111
2 siuntėjas: siunčiu:   2222
3 siuntėjas: siunčiu:    333
4 siuntėjas: siunčiu:     44
5 siuntėjas: siunčiu:      5
**** 1 siuntėjas atsiuntė:  11111
**** 2 siuntėjas atsiuntė:   2222
**** 3 siuntėjas atsiuntė:    333
**** 4 siuntėjas atsiuntė:     44
**** 5 siuntėjas atsiuntė:      5
Programa baigė darbą.
```

Kitos LP priemonės: užraktai (*locks*)

Du užraktų tipai: `sync.Mutex` ir `sync.RWMutex`

```
var l sync.Mutex
var a string
func f() {
  a = "hello, world"
  l.Unlock()
}
func main() {
  l.Lock()
  go f()
  l.Lock()
  print(a)
}
```

Klausimai pakartojimui

1. Kam skirta Go kalba?
2. Su kokia kita kalba gali būti lyginama Go kalba?
3. Kiek reikšmių gali gražinti Go kalbos funkcija?
4. Kurioje Go kalbos versijoje atsirado lygiagrečiojo programavimo priemonės?
5. Kokiu būdu užrašomi Go kalbos procesai?
6. Kokiu būdu komunikuoja Go procesai?
7. Kokiu būdu sinchronizuojami Go procesai?