<01/17/23>

# Go 1.22 range over func/ range over int

**Takuma Shibuya**

CyberAgent Inc.

sivchari

## Self introduction

- Takuma Shibuya
  - sivchari

- CIU
  - AKE

- Go Next Experts

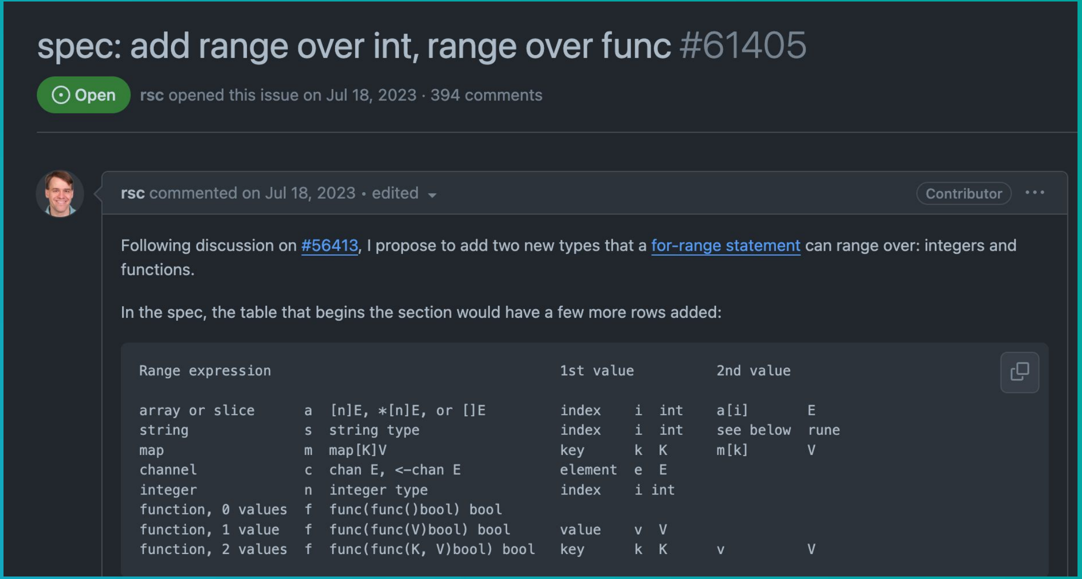
- Go Conference Host

SECTION ONE

## What is “range over specification”

## range over int/func is proposed by rsc

- 61405
- CL



spec: add range over int, range over func #61405

Open rsc opened this issue on Jul 18, 2023 · 394 comments

rsc commented on Jul 18, 2023 • edited

Contributor

Following discussion on #56413, I propose to add two new types that a for-range statement can range over: integers and functions.

In the spec, the table that begins the section would have a few more rows added:

```
Range expression                                  1st value          2nd value

array or slice      a  [n]E, *[n]E, or []E        index    i  int    a[i]       E
string              s  string type                index    i  int    see below  rune
map                 m  map[K]V                    key      k  K      m[k]       V
channel             c  chan E, <-chan E           element  e  E
integer             n  integer type               index    i int
function, 0 values  f  func(func()bool) bool
function, 1 value   f  func(func(V)bool) bool     value    v  V
function, 2 values  f  func(func(K, V)bool) bool  key      k  K      v          V
```

SECTION TWO

## What is “range over int”

## Go1.22 makes two changes to “for” loops

- **Each iteration of the loop creates new variables**
  - **build : go build -gcflags=all=-d=loopvar=2**
  - **test: bisect -compile=loopvar go test**

- **“for” loops may now range over integers**

## The specifications change in Go1.22

```
Range expression                          1st value          2nd value

array or slice  a  [n]E, *[n]E, or []E    index    i  int    a[i]       E
string          s  string type            index    i  int    see below  rune
map             m  map[K]V                key      k  K      m[k]       V
channel         c  chan E, <-chan E       element  e  E
integer         n  integer type I         value    i  I
```

**Example:**

```
for i := range 10 {

      println(i)
}
```

**Output:**

```
0
1
2
3
.
.
9
```

SECTION THREE

## What is “range over func”

## Go1.22 includes a preview of language change Go team is considering for a future version of Go

**Building with GOEXPERIMENT=rangefunc enables this feature**

- GOEXPERIMENT=rangefunc go install my/program
- GOEXPERIMENT=rangefunc go build my/program
- GOEXPERIMENT=rangefunc go run my/program
- GOEXPERIMENT=rangefunc go test my/program

## The specifications change in Go1.22 (+ range over func)

| Range expression | | | 1st value | | | 2nd value | |
|---|---|---|---|---|---|---|---|
| array or slice | a | [n]E, *[n]E, or []E | index | i | int | a[i] | E |
| string | s | string type | index | i | int | see below | rune |
| map | m | map[K]V | key | k | K | m[k] | V |
| channel | c | chan E, <-chan E | element | e | E | | |
| integer | n | integer type | index | i | int | | |
| function, 1 value | f | func(func(V)bool) bool | value | v | V | | |
| function, 2 values | f | func(func(K, V)bool) bool | key | k | K | v | V |

This will allow import of the experimental package iter which exports types

- type Seq[V any] func(yield func(V) bool)
- type Seq2[K, V any] func(yield func(K, V) bool)

And helper functions

- func Pull[V any](sec Sec[V]) (next func() (V, bool), stop func())
- func Pull2[K, V any](seq Seq2[K, V]) (next func(K, V, bool), stop func())

## With GOEXPERIMENT=range func enabled, following range expression will iterate.

```
// f has type Seq[V], v has type V
for v := range f { ... }

// f has Seq2[K, V], k and v have types K and V
for k, v := range f { ... }
```

## Simple case

```
package slices

func Reverse[E any](v []E) func(func(int, E) bool) {
	return func(yield func(int, E) bool) {
		for i := len(s) - 1; i >= 0; i– {
			if !yield(i, s[i]) { return }
		}
		return
	})
}
```

## Simple case

```
package main

func main() {
	s := []string{ “hello”, “world” }
	for i, v := range slices.Reverse(s) {
		fmt.Println(i, v)
	}
}
```

## This program is translated like this by Go-compiler
## This translation is done in rewrite.go

```
slices.Reverse(s)(func(i int, v string) bool {
	fmt.Println(i, v)
	return true
})
```

**The “return true” at the end of the body is the implicit “continue” at the end of the loop body**

**An explicit “continue” would translate to “return true”**

**A “break statement” would translate to “return false”**

## Why are yield functions limited to at most two arguments ?

People may report a bug about the compiler when it crashes

Now, go/ast and go/parser only represent up to two range values and there aren't legitimate, strong reasons to support three or more

The simplest choice is to stop at two and leave those packages unchanged, but if Go team find a strong reason in the future, they will reconsider about the limit

## SUMMARY

**Go1.22 supports new “for” features**

**Go team prepares tools and flags to migrate from Go1.21 to Go1.22**

**range over func is still in progress, but you can enable with GOEXPERIMENT=rangefunc**

**For more detail, you can see here**