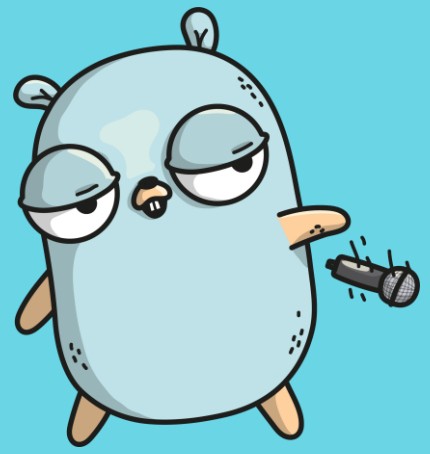


GENERICS

Mehdi Teymorian

Software Engineer @ Snapp

B.Sc. @ Shahid Beheshti

What’s new?

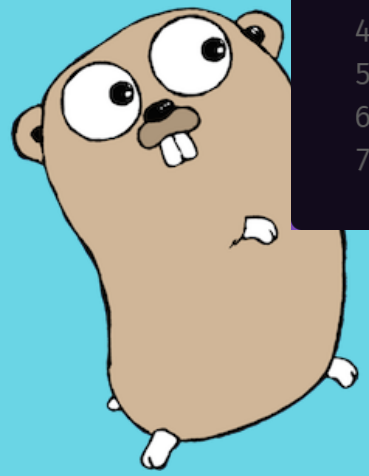
Limitation

Conclusion

Type Parameter

```go
func minInt(first, second int) int {
    if first < second {
        return first
    }
    return second
}

func minFloat64(first, second float64) float64 {
    if first < second {
        return first
    }
    return second
}
```

Type Parameter

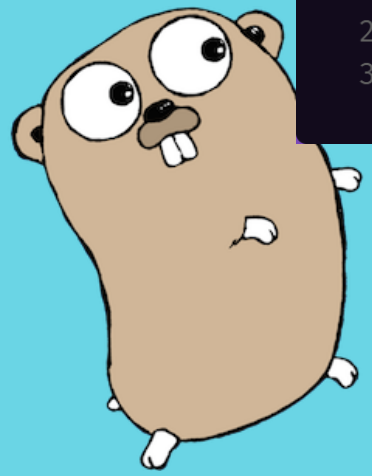


```go
func Min[T int | float64](first, second T) T {
    if first < second {
        return first
    }

    return second
}
```

Type Parameter

```go
import "fmt"

func CallTypeParameter() {
    fmt.Println(Min[int](2, 3))
    fmt.Println(Min[float64](2.3, 1.3))

    fmt.Println(Min(2, 3))
    fmt.Println(Min(2.3, 1.3))
}
```

Multi Type Parameter



```go
func toComplex[T int | float64, P int | float64](a T, b P) complex128 {
    return complex(float64(a), float64(b))
}
```

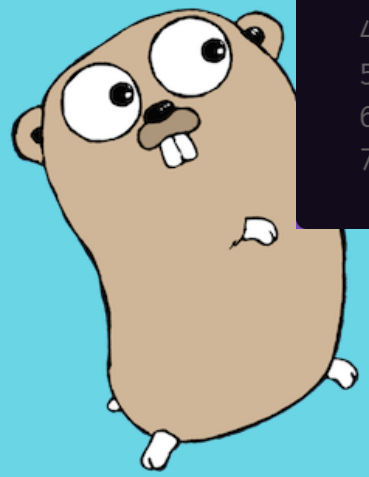
Multi Type Parameter

```go
import "fmt"

func CallMultiTypeParameters() {
    x := 3.2
    y := 2
    fmt.Println(toComplex(x, y))
}
// output: (3.2+2i)
```

Interface Constraint

```go
type Number interface {
    int | float64
}

func Max[T Number](first, second T) T {
    if first > second {
        return first
    }

    return second
}
```

Interface Constraint



```go
import "fmt"


func CallInterfaceConstraint() {
    fmt.Println(Max(1, 2))
    fmt.Println(Max(1.2, 52.2))
}
```

Base Type Parameter

```go
type BaseNumber interface {
    ~int | ~float64
}


func IsEqual[T BaseNumber](first, second T) bool {
    return first == second
}
```

Base Type Parameter

```go
import "fmt"

type AnotherInt int

func CallBaseType() {
    var b AnotherInt = 2
    var a AnotherInt = 2

    fmt.Println(IsEqual(a, b))
}
// output: true
```

Another Example

```go
import "fmt"

type Age int
type Name string

func (a Age) String() string {
    return fmt.Sprintf("Age: %d", a)
}

func (n Name) String() string {
    return "Name: " + string(n)
}

type Stringer interface {
    Age | Name

    String() string
}

func PrintStringer[T Stringer](input T) {
    fmt.Println(input.String())
}
```

Another Example

```go
func CallComplexExample() {
    var age Age = 60
    var name Name = "Mehdi"
    PrintStringer(age)
    PrintStringer(name)
}
//output:
//Age: 60
//Name: Mehdi
```

Any Keyword

```go
func Print[T any](items ...T) {
    for _, item := range items {
        fmt.Println(item)
    }
}
```

Any Keyword

```go
import "fmt"

func CallAny() {
    items := []any{"a", 1, 1.2, false}
    numbers := []int{1, 2, 3, 4, 5}

    Print(items)
    Print(numbers)
}
// output:
//[a 1 1.2 false]
//[1 2 3 4 5]
```

Comparable Keyword

```go
func IsArrayEqual[T comparable](first, second []T) bool {
    // assumption: both arrays are the same length to avoid complexity in code.
    for i := range first {
        if first[i] ≠ second[i] {
            return false
        }
    }
    return true
}
```

Comparable Keyword

```go
import "fmt"

func CallComparable() {
    x := []int{1, 2, 3, 4}
    y := []int{1, 2, 3, 4}

    fmt.Println(IsArrayEqual(x, y))
}
// output: true
```

Generic Struct

```go
type Box[T any] struct {
    Items []T
}

func (b Box[T]) Size() int {
    return len(b.Items)
}

func (b *Box[T]) Add(item T) {
    b.Items = append(b.Items, item)
}

func (b Box[T]) Foreach(consumer func(index int, each T)) {
    for i, item := range b.Items {
        consumer(i, item)
    }
}

```

Generic Struct

```go
type Box[T any] struct {
    Items []T
}

func (b Box[T]) Size() int {
    return len(b.Items)
}

func (b *Box[T]) Add(item T) {
    b.Items = append(b.Items, item)
}

func (b Box[T]) Foreach(consumer func(index int, each T)) {
    for i, item := range b.Items {
        consumer(i, item)
    }
}

```

```go
1 type Box[T any] []T
```

Generic Struct

```go
import "fmt"

func CallGenericStruct() {
    box := Box[int]{Items: []int{1, 2, 3, 45}}

    fmt.Printf("size: %d\n", box.Size())
    box.Add(2)
    fmt.Printf("size after add: %d\n", box.Size())

    box.Foreach(func(index int, item int) {
        fmt.Printf("index: %d, item: %d\n", index, item)
    })
}
// output:
//size: 4
//size after add: 5
//index: 0, item: 1
//index: 1, item: 2
//index: 2, item: 3
//index: 3, item: 45
//index: 4, item: 2
```

Generic Channels

```go
func Merge[T any](channel1, channel2 chan T) chan T {
    mediator := make(chan T)
    go func() {
        for {
            select {
            case item1 := <-channel1:
                mediator <- item1
            case item2 := <-channel2:
                mediator <- item2
            }
        }
    }()

    return mediator
}
```

Generic Channels

```go
func CallGenericChannel() {
    length := 5

    channel1, channel2 := make(chan int), make(chan int)
    mediator := Merge(channel1, channel2)

    go func() {
        for i := 0; i < length; i++ {
            channel1 <- i + 10
            channel2 <- i + 100
            time.Sleep(1 * time.Second)
        }
        close(mediator)
    }()

    for each := range mediator {
        fmt.Println(each)
    }

    fmt.Println("The End")
}
```

Constraints (Exp)

```go
import "golang.org/x/exp/constraints"


// Reduce
// other constraints: integer, complex, float, signed, unsigned
func Reduce[T constraints.Ordered](items []T) T {
    result := *new(T)
    for _, item := range items {
        result += item
    }

    return result
}
```

Constraints (Exp)

```go
import "fmt"


func CallConstraints() {
    numbers := []int{1, 2, 3, 4, 5}
    floats := []float64{1.2, 4.2}

    fmt.Println(Reduce(numbers))
    fmt.Println(Reduce(floats))
}
// output:
// 15
// 5.3
```

Slices & Maps (Exp)

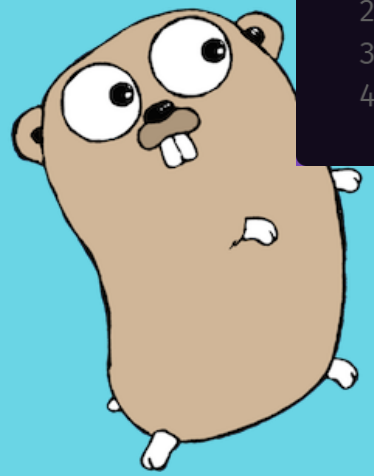
```go
import (
    "fmt"
    "golang.org/x/exp/slices"
)

func CallSliceX() {
    // operations: sort, contain, equal, index

    numbers := []int{1, 4, 6, 12, 34, 67, 23, 97}
    slices.Sort(numbers)
    position := slices.BinarySearch(numbers, 23)
    fmt.Println(numbers)
    fmt.Println(position)
}
// output:
//[1 4 6 12 23 34 67 97]
//4
```

What’s new?

Limitation

Conclusion

✕ Declare Generic Method



```go
func (s AnyStruct) PrintWithLabel[T any](label T) {
    fmt.Println(label)
}
// compile output: syntax error: method must have no type parameters
```

✕ Declare Method for Generic Struct

```go
type RandomStruct[T any] struct {
    Item T
}

func (s RandomStruct) String() string {
    return "name of struct"
}
// compile output: cannot use generic type RandomStruct[T any] without
instantiation
```

✕ Access Struct Fields

```go
type Shape struct {
    width, height int
}

func (s Shape) Area() int {
    return s.width * s.height
}

func CompareShape[T Shape](f, s T) T {
    if f.Area() > s.Area() {
        return f
    }

    return s
}
// compile output: f.Area undefined (type T has no field or method Area)
// same for accessing fields
```

✕ Dec. Type Inside Generic Function

```go
import "fmt"

func Print[T any](something T) {
    type name string

    person := name("mehdi")

    fmt.Println(person)
    fmt.Println(something)
}

// compile output: type declarations inside generic functions are not currently supported
```

✕ Embedded Unnamed Type Parameter

```go
type RandomStruct[T any] struct {
    T
}
// compile output: embedded field type cannot be a (pointer to a) type parameter
```

✕ Union Element With Method

```go
import "fmt"

type Number interface {
    int | int8 | int16 | int32

    String() string
}

func ToggleSign[T int64 | Number](anyNumber T) T {
    return -1 * anyNumber
}

func CallUnionElementLimitation() {
    fmt.Println(ToggleSign(2))
}
// compile output: cannot use generics-go/limitation.Number in union (generics-go/limitation.Number contains methods)
```

✕ real(), imag(), complex()

```go
import "fmt"

func separate[T complex64 | complex128](number T) (float64, float64) {
    return real(number), imag(number)
}

func CallComplexNumber() {
    a := 2 + 3i
    fmt.Println(separate(a))
}
// compile output:
// number (variable of type T constrained by complex64|complex128) not supported as argument to real for go1.18 (see issue #50937)
// number (variable of type T constrained by complex64|complex128) not supported as argument to imag for go1.18 (see issue #50937)

```

What’s new?

Limitation

Conclusion

Conclusion

- Performance is approximately the same
- Implementation may change
- There are proposals including language change
- Light use of generic is OK
- Go 2.x might be a better version for using generics in production

Resources

- Go 1.18 Release Notes
- Type Parameters Proposal
- Go Official Generics Tutorial
- GitHub Repository

Thank you for Listening

 teymorian

 mehditeymorian

 mehdi-teymorian

 mohammadmehdi.teymorian@snapp.cab