



# 谈谈 Go 泛型

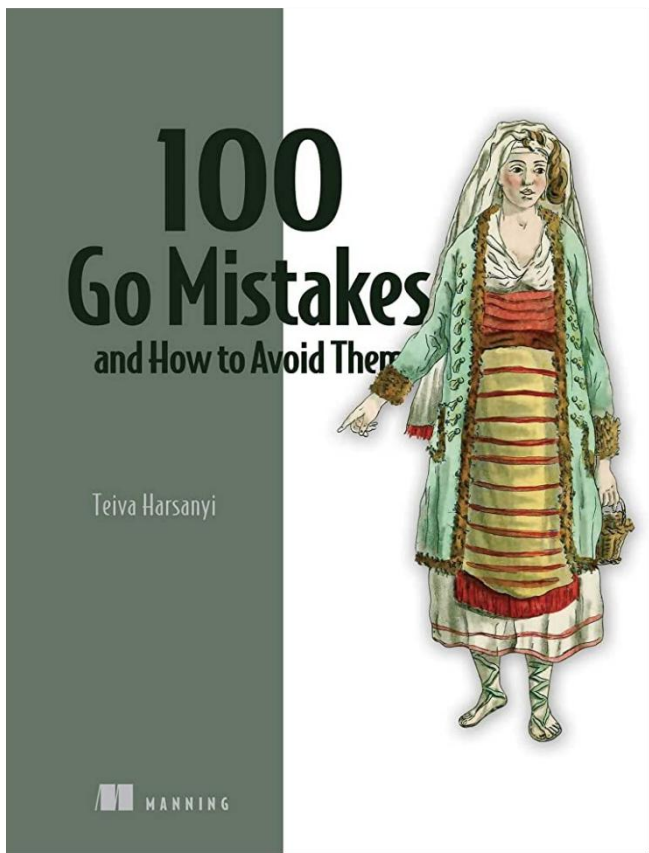


鸟窝 (晁岳攀)

微服务框架 rpcx 作者

# 个人介绍

微服务框架rpcx作者，百度资深工程师



《100个Go语言错误及避坑指南》，译者之一。即将出版



《深入理解Go并发编程》，作者。封面占位用。即将出版



# 目 录

回顾 Go 泛型

01

Go 泛型陷阱

02

最佳实践

03

改造 Go 项目

04

Top N Go 泛型库

05

第一部分

# 回顾 Go 泛型



# 泛型类型和泛型函数

## 泛型类型

类型参数

类型约束

```
type List[T any] struct {  
    next *List[T]  
    value T  
}
```

## 泛型函数

类型参数

类型约束

```
func min[T ~int|~float64](x, y T) T {  
    if x < y {  
        return x  
    }  
    return y  
}
```

## 泛型类型的方法

```
func (l *List[T]) Len() int { ... }
```

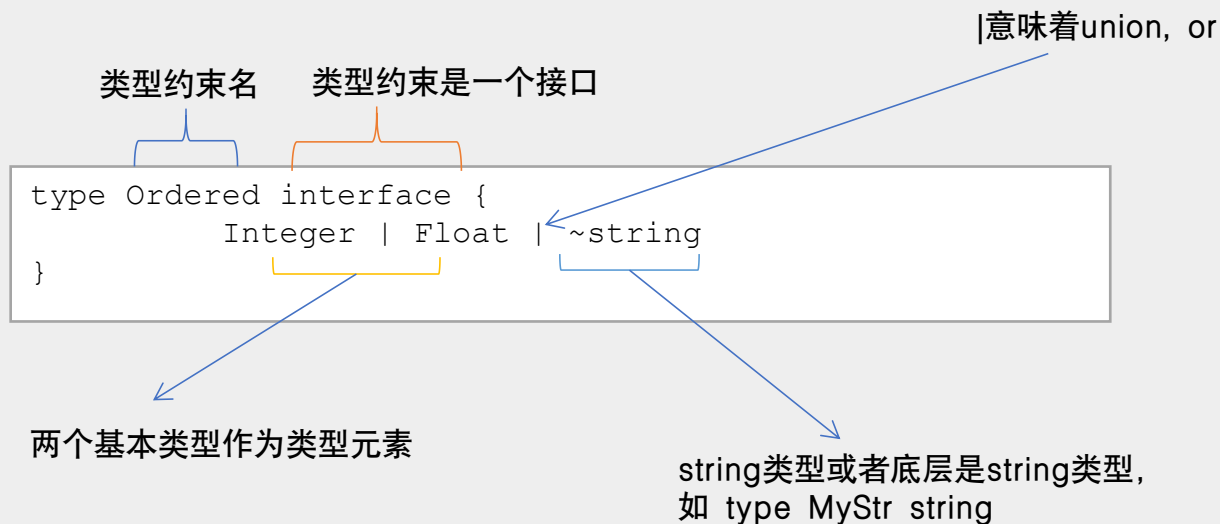
## 泛型方法?

```
func (l *List) Len[V any]() int { ... }
```

# 类型约束和接口

- <go 1.18 interface: an interface type specifies method set called its interface.
- >=go1.18 interface: an interface type defines a type set.

类型约束是一个接口，它为相应的type parameters 定义一组允许的type arguments, 并控制该type parameters 的值所支持的操作。



# 类型约束和接口

```
type Int interface {
    int
}
type AllInt interface {
    int | int8 | int16 | int32 | int64
}
type Signed interface {
    ~int | ~int8 | ~int16 | ~int32 | ~int64
}
type SignedToString interface {
    ~int | ~int8 | ~int16 | ~int32 | ~int64

    ToString() string
}
type Empty interface {
    int
    float64
}
type ReadWriterCloser interface {
    *os.File | *net.TCPConn | *net.UDPConn | *net.UnixConn | *net.IPConn

    io.Reader
    io.Writer
    io.Closer
}
```

**any**

**comparable**

# 更复杂的定义

```
type (  
  S1[P any] struct{}  
  S2[S interface{ ~[]byte | string }] struct{}  
  S3[S ~[]E, E any] struct{}  
  S4[P S1[int]] struct{}  
  S5[_ any] struct{}  
  S6[_ any, _ io.Reader] struct{}  
  S7[K any, _ io.Reader, V any] struct{}  
  S8[K, _ any, S, T S4[S1[int]]] struct{}  
  S9[K int, T map[K]any] struct{}  
)  
type MyList[T any] List[T]  
type MyList2 List[int]  
type MyMap[K comparable, V any] map[K]V  
type MyMap2[K comparable] map[K]int  
  
type MyFunc[T constraints.Ordered, V any] func(T) V  
  
func (l *List[T]) Len() int {  
  ...  
}  
  
func (l *List[V]) Size() int {  
  ...  
}
```



# 泛型的实现

<https://github.com/golang/proposal/blob/master/design/generics-implementation-dictionaries-go1.18.md>

 generics-implementation-dictionaries-go1.18.md	design/generics-implementation-dictionaries-go1.18: add missing period	last year
 generics-implementation-dictionaries.md	design: add GC shape hybrid design doc for the generics implementation	3 years ago
 generics-implementation-gcshape.md	design: fix typo in generics implementation document	last year
 generics-implementation-stenciling.md	design: add GC shape hybrid design doc for the generics implementation	3 years ago

# 泛型的实现

```
main.  
main.main  
runtime.main.func1  
runtime.main.func2  
main.main  
file: /Users/smallnest/workspace/talk-about-go-generics/main.go  
MOVW 16(R28), R16  
CMP R16, RSP  
BLS 65(PC)  
MOVW.W R30, -112(RSP)  
MOVW R29, -8(RSP)  
SUB $8, RSP, R29  
STP (ZR, ZR), 88(RSP)  
ORR $1, ZR, R0  
CALL runtime.convT64(SB)  
ADRP 167936(PC), R1  
ADD $1504, R1, R1  
MOVW R1, 88(RSP)  
MOVW R0, 96(RSP)  
ADRP 700416(PC), R27  
MOVW 1352(R27), R1  
ADRP 245760(PC), R0  
ADD $3208, R0, R0  
ADD $88, RSP, R2  
ORR $1, ZR, R3  
MOVW R3, R4  
CALL fmt.Fprintln(SB)  
STP (ZR, ZR), 72(RSP)  
MOVW $46607182418800017408, R0  
CALL runtime.convT64(SB)  
ADRP 167936(PC), R1  
ADD $1440, R1, R1  
MOVW R1, 72(RSP)  
MOVW R0, 80(RSP)  
ADRP 700416(PC), R27  
MOVW 1352(R27), R1  
ADRP 245760(PC), R0  
ADD $3208, R0, R0  
ADD $72, RSP, R2  
ORR $1, ZR, R3  
MOVW R3, R4  
CALL fmt.Fprintln(SB)  
ADRP 0(PC), R0  
ADD $3424, R0, R0  
ORR $1, ZR, R1  
ADRP 126976(PC), R2  
ADD $2680, R2, R2  
MOVW R1, R3  
CALL runtime.cmpstring(SB)  
STP (ZR, ZR), 56(RSP)  
CMP $0, R0  
ADRP 0(PC), R1  
ADD $3424, R1, R1  
ADRP 126976(PC), R2  
ADD $2680, R2, R2  
CSEL LT, R1, R2, R0  
/Users/smallnest/workspace/talk-about-go-generics/main.go  
7 )  
8  
9 func min[T constraints.Ordered](x, y T) T {  
10 if x < y {  
11 return x  
12 }  
13 return y  
14 }  
15  
16 func main() {  
17 fmt.Println(min(1, 2))  
18 fmt.Println(min(1.0, 2.0))  
19 fmt.Print(min("a", "b"))  
20 }  
21  
/Users/smallnest/sdk/gotip/src/fmt/print.go  
269 // Spaces are added between operands when neither is a string.  
270 // It returns the number of bytes written and any write error encountered.  
271 func Print(a ...any) (n int, err error) {  
272 return Fprint(os.Stdout, a...)  
273 }  
274  
275 // Sprint formats using the default formats for its operands and returns the resulting  
311 // Spaces are always added between operands and a newline is appended.  
312 // It returns the number of bytes written and any write error encountered.  
313 func Println(a ...any) (n int, err error) {  
314 return Fprintln(os.Stdout, a...)  
315 }  
316  
317 // Sprintln formats using the default formats for its operands and returns the resulti
```

# 泛型的实现

The screenshot shows the lensm IDE interface. At the top, a terminal window displays the command `gotip build -gcflags="-N -l" .` and `lensm ./talk-about-go-generics`. Below the terminal, the main editor area is split into two panes. The left pane shows the Go source code for `main.main`, with a sidebar on the far left listing symbols like `main_min.go.shape.float64`. The right pane shows the corresponding assembly code, with colored lines connecting the assembly instructions to their corresponding source code lines. The assembly code includes instructions like `MOVD $1, ZR, R1`, `CALL main.min[go.shape.int](SB)`, and `JMP 2(PC)`.

- [loov.dev/lensm](https://loov.dev/lensm)
- <https://godbolt.org/>

# 标准库中的使用

- **atomic**

**type Pointer**

added in go1.19

```
type Pointer[T any] struct {  
    // contains filtered or unexported fields  
}
```

A Pointer is an atomic pointer of type \*T. The zero value is a nil \*T.

**func (\*Pointer[T]) CompareAndSwap**

added in go1.19

```
func (x *Pointer[T]) CompareAndSwap(old, new *T) (swapped bool)
```

- **exp**

- [github.com/golang/exp/constraints](https://github.com/golang/exp/constraints)
- [github.com/golang/exp/maps](https://github.com/golang/exp/maps)
- [github.com/golang/exp/slices](https://github.com/golang/exp/slices)
- maps
- slices
- cmp

- **arena**

```
func New[T any](a *Arena) *T {  
    return runtime_arena_arena_New(a.a, reflectlite.TypeOf((*T)(nil))).(*T)  
}
```

# 不甚满意的地方

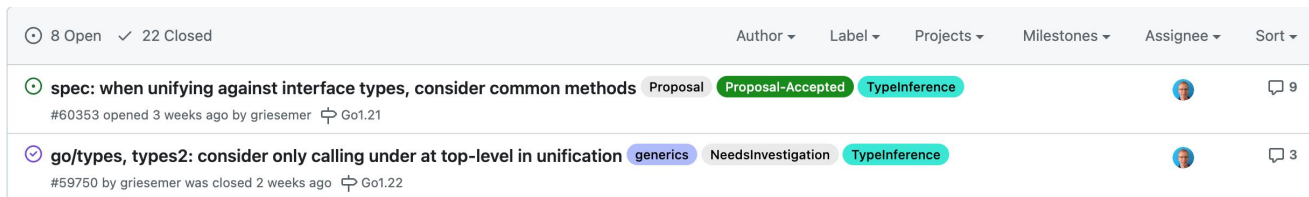
- 使用类型集的共同字段 (Go1.22)

<https://github.com/golang/go/issues/48522>

- 实现泛型方法

<https://github.com/golang/go/issues/49085>

- 类型推断更智能



A screenshot of the GitHub issues page for the Go 1.22 release. The page shows two issues related to type inference and generics. The first issue, #60353, is titled "spec: when unifying against interface types, consider common methods" and is labeled as "Proposal-Accepted" and "TypeInference". It was opened 3 weeks ago by griesemer. The second issue, #59750, is titled "go/types, types2: consider only calling under at top-level in unification" and is labeled as "generics", "NeedsInvestigation", and "TypeInference". It was closed 2 weeks ago by griesemer. The page also shows filters for 8 Open and 22 Closed issues, and various sorting and filtering options.

Issue ID	Title	Labels	Status	Author	Comments
#60353	spec: when unifying against interface types, consider common methods	Proposal, TypeInference	Proposal-Accepted	griesemer	9
#59750	go/types, types2: consider only calling under at top-level in unification	generics, NeedsInvestigation, TypeInference	Closed	griesemer	3

- 一些被Hold的提案

- proposal: context: add generic Key type
- proposal: spec: allow type parameters in methods
- proposal: spec: allow values of type comparable

# 性能争议

---

- <https://planetscale.com/blog/generics-can-make-your-go-code-slower>
- <https://www.infoq.com/news/2022/04/go-generics-performance/>
- <https://github.com/golang/go/issues/54238>
- <https://levelup.gitconnected.com/go-generics-holistic-thoughts-on-performance-fc6688fb414b?gi=66192a253f10>
- <https://www.dolthub.com/blog/2022-04-01-fast-generics/>

第二部分

# Go 泛型陷阱



# 陷阱一：无效的近似元素

```
type Error struct {
    Code int
    Error string
}

type I0 interface {
    ~Error // struct
    ~error // interface
}

type Error2 = struct {
    Code int
    Error string
}

type I00 interface {
    ~Error2 // OK
}
```

- Type literals, such as []byte or struct{ f int }.
- Most predeclared types, such as int or string (but not error).
- Using ~T is not permitted if T is a type parameter or if T is an interface type.



## 陷阱二：类型参数在错误的位置

```
type T1[P any] P

type T2[P any] struct{ *P }

func I1[K any, V interface{ K }]() {
}

func I2[K any, V interface{ int | K }]() {
}
```

## 陷阱三：重叠

```
func I4[K any, V interface{ int | ~int }]() {  
}  
  
type MyInt int  
  
func I5[K any, V interface{ int | MyInt }]() {  
}  
  
type MyInt2 = int  
  
func I6[K any, V interface{ int | MyInt2 }]() {  
}  
  
func I7[K any, V interface{ int | any }]() {  
}
```

# 陷阱四：约束用作传统接口

```
type IntStr interface {
    int | string
}

func add[T IntStr](a, b T) T {
    return a + b
}

func add1(a, b IntStr) IntStr { // 回不去了，没办法在当做普通接口
    return a + b
}
```

# 陷阱五：接口在union中

```
func I11[K interface {
    io.Reader
    io.Writer
}]() {
}

func I12[K interface {
    io.Reader | io.Writer
}]() {
}

func I13[K interface {
    io.Reader | os.File
}]() {
}

func I14[K interface {
    any | os.File
}]() {
}

func I12_1[K interface {
    interface{ foo() } | int
}]() {
}
```

# 陷阱六：comparable在union中

```
func I15[K interface {
    comparable
}] () {
}

func I16[K interface {
    error
    comparable
}] () {
}

func I17[K interface {
    comparable | os.File
}] () {
}
```

# 陷阱七：递归类型

```
type Tree[T any] struct {  
    Left  *Tree[T]  
    V     Value[T, int]  
    Right *Tree[T]  
}  
  
type Value[T any, V any] struct {  
    Hodler Tree[T] // *Tree[T]  
    Value  V  
}
```

# 陷阱八：nil比较

---

```
func ZeroValue0[T any](v T) bool {  
    return v == nil  
}  
  
func interfaceIsNil(v any) bool {  
    return v == nil  
}
```

```
func ZeroValue[T comparable](v T) bool {  
    var t T  
    return v == t  
}
```

# 陷阱九：类型推断

```
func Scale[E constraints.Integer](s []E, c E) []E {
    r := make([]E, len(s))
    for i, v := range s {
        r[i] = v * c
    }
    return r
}
```

```
type Point []int

func (p Point) String() string {
    return ""
}

func ScaleAndPrint(p Point) {
    r := Scale(p, 2)
    fmt.Println(r.String())
}
```

```
func Scale1[S ~[]E, E constraints.Integer](s S, c E) S
{
    r := make(S, len(s))
    for i, v := range s {
        r[i] = v * c
    }
    return r
}
```



# 陷阱十：类型不满足

```
type Int1 interface {
    int16 | int32 | int64
}

func f[T Int1](x T) {
    g(x)
    h(x) // T不满足Int3
}

type Int2 interface {
    int16 | int32 | int64
}

func g[T Int2](x T) {
}

type Int3 interface {
    int16 | int32
}

func h[T Int3](x T) {
}

func test() {
    f(int16(0))
}
```

# 陷阱十一：模棱两可的代码

```
type (  
    P struct{}  
    C struct{}  
  
    TT0 [P * C]struct{} // P*C  
    TT1 [P(C)]struct{}  // P(C)  
  
    TT2[P interface{ *C }] struct{}  
    TT3[P interface{ C  }] struct{}  
)
```

## 陷阱十二：零值

```
func Zero1[T any]() T {
    return *new(T)
}

func Zero2[T any]() T {
    var t T
    return t
}

func Zero3[T any]() (t T) {
    return
}
```

- Use `var zero T`, as above, which works with the existing design but requires an extra statement.
- Use `*new(T)`, which is cryptic but works with the existing design.
- For results only, name the result parameter, and use a naked `return` statement to return the zero value.
- Extend the design to permit using `nil` as the zero value of any generic type (but see [issue 22729](#)).
- Extend the design to permit using `T{}`, where `T` is a type parameter, to indicate the zero value of the type.
- Change the language to permit using `_` on the right hand of an assignment (including `return` or a function call) as proposed in [issue 19642](#).
- Change the language to permit `return ...` to return zero values of the result types, as proposed in [issue 21182](#).

# 陷阱十三：泛型方法

```
db, err := database.Connect("...")
if err != nil {
    log.Fatal(err)
}

client := &Client{db}

all, err := client.All[Person](ctx)
if err != nil {
    log.Fatal(err)
}
```

```
package database

type Client struct{ ... }

type Querier[T any] struct {
    client *Client
}

func NewQuerier[T any](c *Client) *Querier[T] {
    return &Querier[T]{
        client: c,
    }
}

func (q *Querier[T]) All(ctx context.Context) ([]T, error) {
    // implementation
}

func (q *Querier[T]) Filter(ctx context.Context, filter ...Filter) ([]T, error) {
    // implementation
}
```

# 陷阱十四： 嵌入T

---

```
type Animal[T any] struct {  
    Val T  
}  
  
type Animal2[T any] struct {  
    T  
}  
  
type Animal3[T any] struct {  
    *T  
}
```

# 陷阱十五: Assignability

```
type Bytes []byte

var x []byte
var y Bytes

func init() {
    x = y // okay
    y = x // okay
}

func f[T Bytes](v T) {
    x = v // okay
    y = v // error
    v = x // okay
    v = y // error
}

func g[T []byte](v T) {
    x = v // okay
    y = v // error
    v = x // okay
    v = y // error
}

func h[T Bytes | []byte](v T) {
    x = v // okay
    y = v // error
    v = x // okay
    v = y // error
}
```

if x's type V or T are type parameters,  
x is assignable to a variable of type T  
if one of the following conditions applies:

- x is the predeclared identifier nil, T is a type parameter, and x is assignable to each type in T's type set.
- V is not a named type, T is a type parameter, and x is assignable to each type in T's type set.
- V is a type parameter and T is not a named type, and values of each type in V's type set are assignable to T.

# 陷阱十六：取指针

---

```
func ptr[T any](v T) *T {
    return &v
}

func TestPtr(t *testing.T) {
    var x int
    var y = ptr(x) // 生成一个指向 x类型 的指针，不是指向x的指针
    *y = 10

    assert.Equal(t, 0, x)
}
```

# 陷阱十七： type switch

```
func tswitch[T any](v T) {
    switch v := v.(type) {
    case int:
        println(v)
    case string:
        println(v)
    default:
        println(v)
    }
}

func tswitch2[T any](v T) {
    switch (any)(v).(type) {
    case int:
        println(v)
    case string:
        println(v)
    default:
        println(v)
    }
}
```



# 陷阱十八: type assert

```
func tassert[T any](v T) {
    i := v.(int)
    fmt.Println(i)
}

func tassert2[T any](v T) {
    i := int(v)
    fmt.Println(i)
}

func tassert3[T any](v T) {
    i := (any)(v).(int)
    fmt.Println(i)
}

func tassert4[T constraints.Integer](v T) {
    i := int(v)
    fmt.Println(i)
}
```

# 陷阱十九: comparable vs Ordered

```
func compare[T comparable](a, b T) int {
    if a < b {
        return -1
    } else if a == b {
        return 0
    } else {
        return 1
    }
}

func compare2[T cmp.Ordered](a, b T) int {
    if a < b {
        return -1
    } else if a == b {
        return 0
    } else {
        return 1
    }
}
```

## 陷阱二十： type parameter as RHS

---

```
type Foo[T any] []T
type Bar[T any] T
type Baz[T any] struct{ T }
```

# 陷阱二十一： require pointer methods

```
func Stringify[T fmt.Stringer](s []T) (ret []string) {
    for i := range s {
        ret = append(ret, s[i].String())
    }
    return ret
}

func TestStringify() {
    var s []bytes.Buffer

    var ret []string
    for i := range s {
        ret = append(ret, s[i].String())
    }

    var result = Stringify(s)
    fmt.Println(result)
}
```

第三部分

# 最佳实践



# When using language-defined container types

<https://go.dev/blog/when-generics>

- 啥时候泛型有用?
  - 当使用语言定义的容器类型时
  - 通用目的的数据结构
- 啥时候泛型无用?
  - 不要使用泛型替换接口
  - 当方法的实现不同时，不要使用泛型

如果您发现自己多次编写完全相同的代码，其中副本之间的唯一区别是代码使用不同的类型，请考虑是否可以使用类型参数。

另一种说法是，在注意到要多次编写完全相同的代码之前，应避免使用类型参数。

# 通用的数据结构和算法

## slices

```
func BinarySearch(x []E, target E) (int, bool)
func BinarySearchFunc(x []E, target T, cmp func(E, T) int) (int, bool)
func Clip(s S) S
func Clone(s S) S
func Compact(s S) S
func CompactFunc(s S, eq func(E, E) bool) S
func Compare(s1, s2 []E) int
func CompareFunc(s1 []E1, s2 []E2, cmp func(E1, E2) int) int
func Contains(s []E, v E) bool
func ContainsFunc(s []E, f func(E) bool) bool
func Delete(s S, i, j int) S
func DeleteFunc(s S, del func(E) bool) S
func Equal(s1, s2 []E) bool
func EqualFunc(s1 []E1, s2 []E2, eq func(E1, E2) bool) bool
func Grow(s S, n int) S
func Index(s []E, v E) int
func IndexFunc(s []E, f func(E) bool) int
func Insert(s S, i int, v ...E) S
func IsSorted(x []E) bool
func IsSortedFunc(x []E, cmp func(a, b E) int) bool
func Max(x []E) E
func MaxFunc(x []E, cmp func(a, b E) int) E
func Min(x []E) E
func MinFunc(x []E, cmp func(a, b E) int) E
func Replace(s S, i, j int, v ...E) S
func Reverse(s []E)
func Sort(x []E)
func SortFunc(x []E, cmp func(a, b E) int)
func SortStableFunc(x []E, cmp func(a, b E) int)
```

## maps

```
func Clone(m M) M
func Copy(dst M1, src M2)
func DeleteFunc(m M, del func(K, V) bool)
func Equal(m1 M1, m2 M2) bool
func EqualFunc(m1 M1, m2 M2, eq func(V1, V2) bool) bool
func Keys(m M) []K
func Values(m M) []V
```

## cmp

```
func Compare(x, y T) int
func Less(x, y T) bool
type Ordered
```

## [golang.org/x/exp/constraints](https://golang.org/x/exp/constraints)

```
type Complex
type Float
type Integer
type Ordered
type Signed
type Unsigned
```

# 数据处理

<https://itnext.io/golang-1-18-generics-the-good-the-bad-the-ugly-5e9fa2520e76>

[github.com/samber/lo](https://github.com/samber/lo)

```
func Assign(maps ...map[K]V) map[K]V
func Associate(collection []T, transform func(item T) (K, V)) map[K]V
func Async(f func() A) <-chan A
func Async0(f func()) <-chan struct{}
func Async1(f func() A) <-chan A
func Async2(f func() (A, B)) <-chan Tuple2[A, B]
func Async3(f func() (A, B, C)) <-chan Tuple3[A, B, C]
func Async4(f func() (A, B, C, D)) <-chan Tuple4[A, B, C, D]
func Async5(f func() (A, B, C, D, E)) <-chan Tuple5[A, B, C, D, E]
func Async6(f func() (A, B, C, D, E, F)) <-chan Tuple6[A, B, C, D, E, F]
func Attempt(maxIteration int, f func(index int) error) (int, error)
func AttemptWhile(maxIteration int, f func(int) (error, bool)) (int, error)
func AttemptWhileWithDelay(maxIteration int, delay time.Duration, ...) (int, time.Duration, error)
func AttemptWithDelay(maxIteration int, delay time.Duration, ...) (int, time.Duration, error)
func Batch(ch <-chan T, size int) (collection []T, length int, readTime time.Duration, ok bool) DEPRECATED
func BatchWithTimeout(ch <-chan T, size int, timeout time.Duration) (collection []T, length int, readTime time.Duration, ok bool) DEPRECATED
func Buffer(ch <-chan T, size int) (collection []T, length int, readTime time.Duration, ok bool)
func BufferWithTimeout(ch <-chan T, size int, timeout time.Duration) (collection []T, length int, readTime time.Duration, ok bool)
func ChannelDispatcher(stream <-chan T, count int, channelBufferCap int, ...) []<-chan T
func ChannelMerge(channelBufferCap int, upstreams ...<-chan T) <-chan T DEPRECATED
func ChannelToSlice(ch <-chan T) []T
func Chunk(collection []T, size int) [][]T
func ChunkString(str T, size int) []T
func Clamp(value T, min T, max T) T
func Coalesce(v ...T) (result T, ok bool)
func Compact(collection []T) []T
func Contains(collection []T, element T) bool
```



# 函数式编程

---

[github.com/samber/mo](https://github.com/samber/mo)

- `Option[T]` (Maybe)
- `Result[T]`
- `Either[A, B]`
- `EitherX[T1, ..., TX]` (With X between 3 and 5)
- `Future[T]`
- `IO[T]`
- `IOEither[T]`
- `Task[T]`
- `TaskEither[T]`
- `State[S, A]`

[github.com/OlegStotsky/go-monads](https://github.com/OlegStotsky/go-monads)

- `Maybe[T]`
- `IO[T]`
- `Either[L, R]`
- `State[S, A]`

# 接口 or 泛型?:

---

## 接口

```
func ReadZippedData(r io.Reader) ([]byte, error) {  
    r = snappy.NewReader(r)  
  
    return io.ReadAll(r)  
}
```

## 泛型

```
func ReadZippedData2[T io.Reader](r T) ([]byte, error) {  
    r = snappy.NewReader(r)  
  
    return io.ReadAll(r)  
}
```

# 不同的type argument不同的实现

## 简单例子

```
func Index[T map[int]string | []string](m T, k int) string {  
    return m[k]  
}  
  
func IndexMap[T map[int]string](m T, k int) string {  
    return m[k] // map的查找算法  
}  
  
func IndexSlice[T []string](m T, k int) string {  
    return m[k] // slice查找算法  
}
```

其它例子：DAO layers

# 适合用reflect的场景 encode/json

```
switch t.Kind() {
case reflect.Bool:
    return boolEncoder
case reflect.Int, reflect.Int8, reflect.Int16, reflect.Int32, reflect.Int64:
    return intEncoder
case reflect.Uint, reflect.Uint8, reflect.Uint16, reflect.Uint32, reflect.Uint64, reflect.Uintptr:
    return uintEncoder
case reflect.Float32:
    return float32Encoder
case reflect.Float64:
    return float64Encoder
case reflect.String:
    return stringEncoder
case reflect.Interface:
    return interfaceEncoder
case reflect.Struct:
    return newStructEncoder(t)
case reflect.Map:
    return newMapEncoder(t)
case reflect.Slice:
    return newSliceEncoder(t)
case reflect.Array:
    return newArrayEncoder(t)
case reflect.Pointer:
    return newPtrEncoder(t)
default:
    return unsupportedTypeEncoder
}
```

第四部分

# 改造既有项目

# 改造

---

## struct类型等

- 找出变化项
- 定义泛型类型
- 修改泛型方法

## 函数

- 找出变化项
- 修改泛型函数

# 找出变化项，定义泛型类型

```
import (  
    "container/heap"  
    "fmt"  
)  
  
// An IntHeap is a min-heap of ints.  
type IntHeap []int
```

```
import (  
    "container/heap"  
    "fmt"  
)  
  
// A Heap is a min-heap of Ts.  
type Heap[T cmp.Ordered] []T
```

# 改造函数和方法

```
func (h IntHeap) Len() int      { return len(h) }
func (h IntHeap) Less(i, j int) bool { return h[i] <
h[j] }
func (h IntHeap) Swap(i, j int)      { h[i], h[j] = h[j],
h[i] }

func (h *IntHeap) Push(x any) {
    *h = append(*h, x.(int))
}

func (h *IntHeap) Pop() any {
    old := *h
    n := len(old)
    x := old[n-1]
    *h = old[0 : n-1]
    return x
}
```

```
func (h Heap[_]) Len() int      { return len(h) }
func (h Heap[_]) Less(i, j int) bool { return h[i] <
h[j] }
func (h Heap[_]) Swap(i, j int)      { h[i], h[j] = h[j],
h[i] }

func (h *Heap[T]) Push(x T) {
    *h = append(*h, x)
}

func (h *Heap[T]) Pop() T {
    old := *h
    n := len(old)
    x := old[n-1]
    *h = old[0 : n-1]
    return x
}
```



第五部分

# Top N Go 泛型库



# 几个有特色的泛型库

---

- `samber/lo`
- `samber/do`
- `samber/mo`
- `ugurcsen/gods-generic`
- `deckarep/golang-set`
- `zyedidia/generic`
- `rakeeb-hossain/functools`
- `Code-Hex/go-generics-cache`
- `golobby/orm`
- `repeale/fp-go`
- `nikgalushko/fx`
- `OlegStotsky/go-monads`
- `mattn/go-generics-example`
- `barweiss/go-tuple`

# samber/lo

---

- Filter
- Map
- FilterMap
- FlatMap
- Reduce
- ReduceRight
- ForEach
- Times
- Uniq
- UniqBy
- GroupBy
- Chunk
- PartitionBy
- Flatten
- Interleave
- Shuffle
- Reverse
- Fill
- Repeat
- Associate / SliceToMap
- Drop
- DropRight
- DropWhile
- DropRightWhile
- Reject
- Count
- CountBy
- CountValues
- CountValuesBy
- Subset
- Slice
- Replace
- ReplaceAll
- Compact
- IsSorted
- IsSortedByKey
- Keys
- ValueOr
- Values
- PickBy
- PickByKeys
- PickByValues
- OmitBy
- OmitByKeys
- OmitByValues
- Entries / ToPairs
- FromEntries / FromPairs
- Invert
- Assign (merge of maps)
- MapKeys
- MapValues
- MapEntries
- MapToSlice



谢谢大家