

# GENERICSS

## A GUIDE FOR THE EAGER ENGINEER

# ALEX RIOS

## LIDER SUPREMO

# ATO 1

## COMO CHEGAMOS AQUI?

# ANSIEDADE

# 2010

- ▶ Exceptions
- ▶ Generics

# EXCEPTIONS



- ▶ Runtime Exception = Panic
- ▶ Try / Catch = Recovery

# GENERIC<sup>S</sup>

A.K.A. Glorified type-checked macros.

**2014 - 2018**

**22 EXPERIENCE REPORTS SOBRE GENERICS**

JAN / 2021

43651: spec: add generic programming using type parameters

450+ comments



# TYPE PARAMETERS PROPOSAL

Authors: Ian Lance Taylor & Robert Griesemer



(1778)

Proposal-Accepted



(121)

# O QUE FAZIAMOS ANTES DE 2021?

- ▶ Interface {} 
- ▶ Reflection 禁 
- ▶ Code generation 

**ATO 2  
RECEIO**

# FUD NA COMUNIDADE



Aram Hăvărneanu  
@aramh

The more I use Go, the more I think generics would be a useless misfeature, why do so many people think they need them? #golang

[Traduzir Tweet](#)

(tweets que envelhecem mal)

**POR QUE TANTA  
PREOCUPAÇÃO?**

NÃO DA PARA SENTIR FALTA  
DO QUE NÃO TEVE.

# DOTGO 2015

## FUNCTIONAL GO - FRANCESC CAMPOY.



## Functional Go

- Doable
  - but slower than normal code
  - requires reflection magic

# TRANSTORNO DE JAVA PÓS-TRAUMÁTICO

é uma condição de saúde mental que é desencadeada pela exposição a longos períodos de programação em Java.

– Lider Supremo

# TJPT

- ▶ Variância X Covariância X Contravariância
- ▶ Type erasure
- ▶ Unbounded wildcard instantiation



**#TGIF?  
THANK GOD IT'S FRIDAY**

#TGIG  
THANK GOD IT'S GO

**FALTA DE CONFIANÇA NA MATURIDADE DA  
COMUNIDADE**

# CONTROLE

- ▶ Como isso vai ser usado no meu projeto?
- ▶ As pessoas não vão abusar do uso?

**A RECOMENDAÇÃO É MUITO PERMISSIVA.**

Crie o proprio guideline

# PRIORIDADE

- ▶ Sua empresa
- ▶ Time
- ▶ Go team
- ▶ Sei la quem da internet

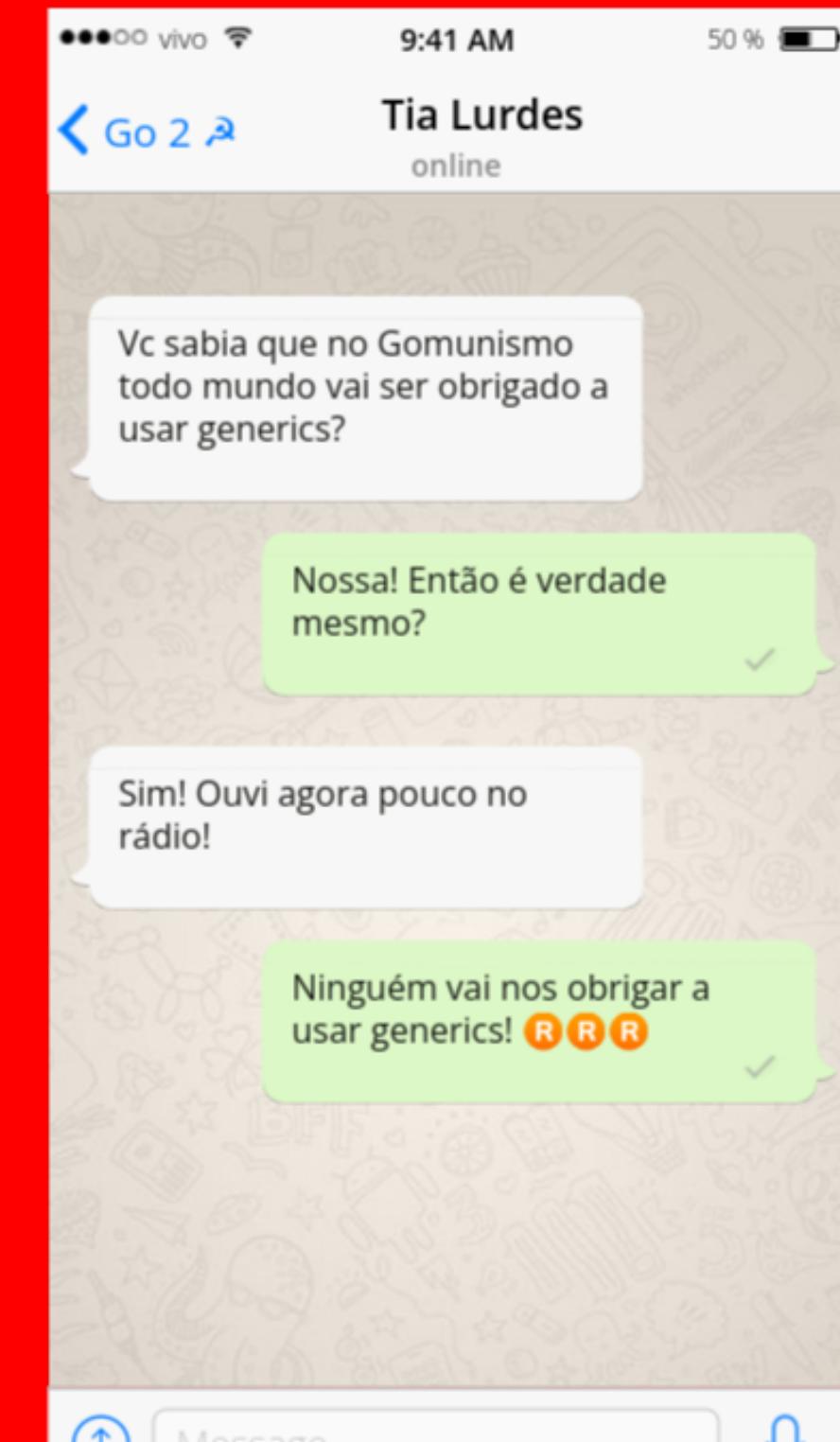
# ABUSO NO USO

- ▶ Vai ser usado em qualquer lugar sem necessidade.
- ▶ Vai destruir o Go (lol?)

# FAKE NEWS



**NO GOMUNISMO TODO  
MUNDO VAI SER  
OBRIGADO A USAR  
GENERICOS.**



GO 2?

# NUNCA

There are no plans for anything called Go 2.

– Ian Lance Taylor

# ATO 3

# INTERFACES & GENERICS

**INTERFACES NÃO ELIMINAM CÓDIGO  
DUPLICADO, APENAS O CONDENSA.**

**TODA TYPE CONSTRAINT É UMA INTERFACE,  
MAS NEM TODA INTERFACE É UMA TYPE  
CONSTRAINT**

# INTERFACES

- ▶ Restringem um comportamento
- ▶ Method Set (conjunto de métodos)

# TYPE CONSTRAINTS

- ▶ Restringem os tipos
- ▶ Type Set (conjunto de tipos)

# TYPE CONSTRAINTS

A interface só é uma Type Constraint a partir do momento que ela declara um type set (conjunto de tipos).

**LET'S PLAY!**

**INTERFACE OU TYPE  
CONSTRAINT**

# ROUND 1

```
type Finder interface {  
    FindByID(id int64) string  
}
```

# ROUND 2

```
type Integer interface {  
    int | int8 | int16 | int32 | int64  
}
```

# ROUND 3

```
type IntegerFinder interface {  
    int~  
    FindByID(id int64) string  
}
```

# FINAL ROUND

Any?

# ATO 4

## PERFORMANCE

**CÓDIGO GENÉRICO É MAIS LENTO QUE MEU  
CÓDIGO ATUAL?**

NAO,  
MAS PODE SER.

# **PLANETSCALE BLOG**

## **GENERICS CAN MAKE YOUR GO CODE SLOWER**

# GO GENERICS

- ▶ Quando?
- ▶ Por quê?
- ▶ Como?

# QUANDO DURANTE A COMPILAÇÃO.

O que torna compilação é mais lenta.

# POR QUÊ?

A penalidade de introduzir a feature poderia afetar 3 pontos:

- ▶ Compilação
- ▶ Programadora
- ▶ Execução

**COMO?**

**Monomorfismo-ish.**

# MONOMORFISMO

Ao contrário do polimorfismo onde o dispatch vai ocorrer em tempo de execução, no Monomorfismo a chamadas são "substituídas" em tempo de compilação.

# STENCILING



Credit: Kim Watson  
via Club Creating Keepsakes

# GCshape Stenciling

## Go 1.18 Implementation of Generics via Dictionaries and Gcshape Stenciling

This document describes the implementation of generics via dictionaries and gcshape stenciling in Go 1.18. It provides more concrete and up-to-date information than described in the [Gcshape design document](#).

The compiler implementation of generics (after typechecking) focuses mainly on creating instantiations of generic functions and methods that will execute with arguments that have concrete types. In order to avoid creating a different function instantiation for each invocation of a generic function/method with distinct type arguments (which would be pure stenciling), we pass a **dictionary** along with every call to a generic function/method. The dictionary provides relevant information about the type arguments that allows a single function instantiation to run correctly for many distinct type arguments.

However, for simplicity (and performance) of implementation, we do not have a single compilation of a generic function/method for all possible type arguments. Instead, we share an instantiation of a generic function/method among sets of type arguments that have the same gcshape.

# PERFORMANCE?

**FACA BENCHMARKS!**  
**PERFÓRMANCE NÃO SE ADIVINHA.**

# ATO 5 GUIDELINES

# QUANDO USAR?

- ▶ Quando estiver usando tipos parametrizados, prefira funções ao uso de métodos.
- ▶ Quando um método tiver a mesma forma para todos os tipos.

Backed by Go team

# QUANDO NÃO USAR?

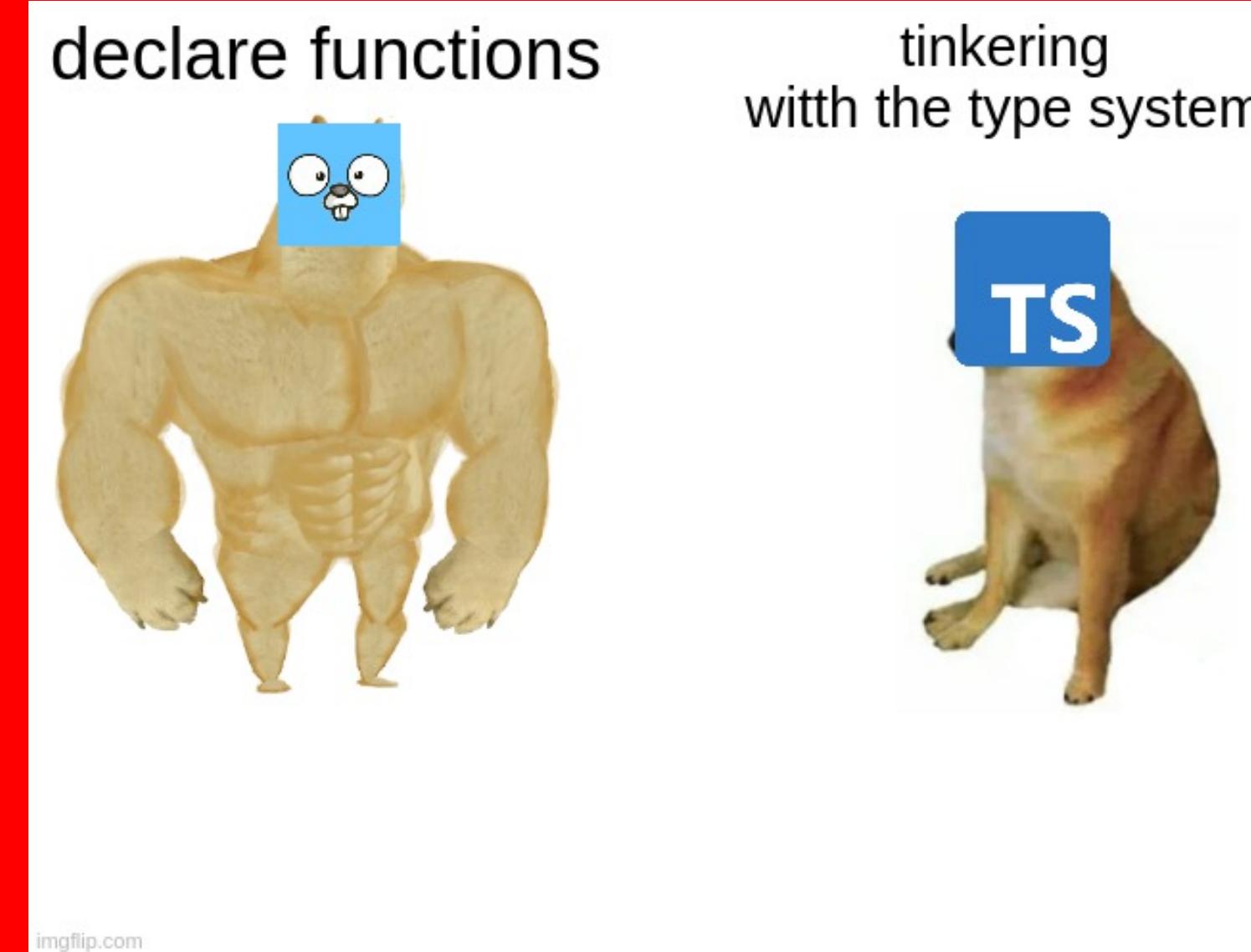
- ▶ Ao apenas chamar um método no argumento de tipo.
- ▶ Quando a implementação para cada tipo for diferente.
- ▶ Quando a operação for diferente para cada tipo, mesmo sem um método.

Backed by Go team

**EVITE**

**BOILERPLATE**

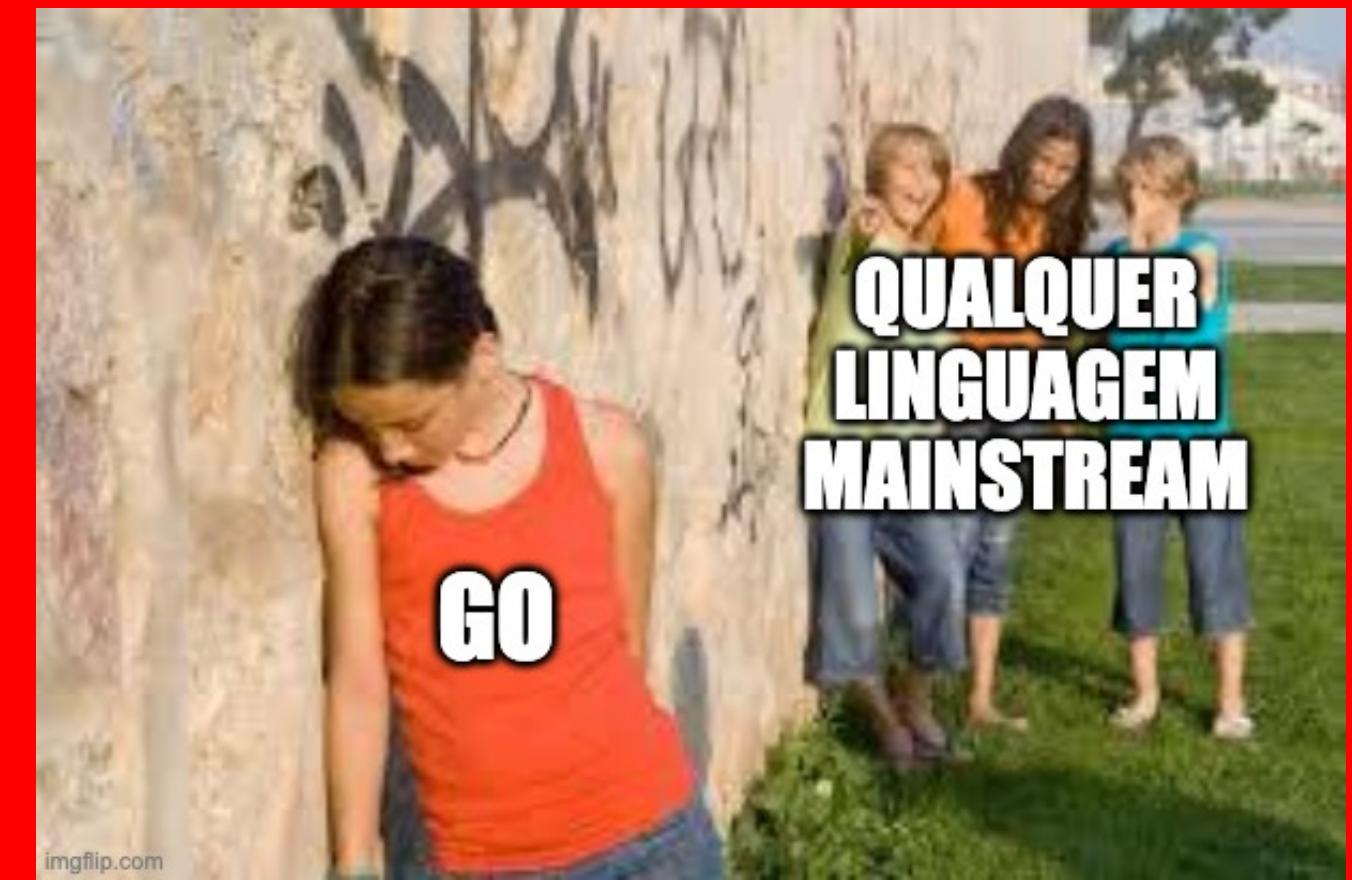
# SE PROGRAMA EM GO CRIANDO FUNÇÕES E NÃO DEFININDO TIPOS.



# APLICAÇÕES COMUÍNS

# SET (CRINGE MODE TRADICIONAL)

```
var myset map[type]struct{}
```



# INTERSECÃO NESSE SET

< 20+ linhas de código ultra repetitivo  
aqui>

- ▶ cria um set para receber os valores da intersecção
- ▶ itera sobre os elementos do set original
- ▶ verifica se um elemento está contido no outro set
- ▶ append no set resultado e retorna

# INTERSEÇÃO COM GENERICS

```
func(s Set[E]) Intersection(s2 Set[E]) Set[E] {  
    result := NewSet[E]()  
    for _, v := ranges.Members() {  
        if s2.Contains(v){  
            result.Add(v)  
        }  
    }  
    return result  
}
```

# NÃO TEM UM LIBZINHA PRA FACILITAR?

Algumas! Por exemplo: <https://github.com/samber/lo>

```
list := []int{1, 2, 3, 4}
isEvenFn := func(x int, _ int) bool {
    return x%2 == 0
}
even := lo.Filter[int](list, isEvenFn)
// []int{2, 4}
```

E MUITO MAIS

Filter Map FilterMap FlatMap Reduce ReduceRight ForEach Times Uniq  
UniqBy GroupBy Chunk PartitionBy Flatten Shuffle Reverse Fill Repeat  
RepeatBy KeyBy Associate SliceToMap Drop DropRight DropWhile  
DropRightWhile Reject Count CountBy Subset Slice Replace ReplaceAll  
Compact IsSorted IsSortedByKey

**GO WAY?  
IDIOMÁTICO?**

# EU NÃO QUERO INTRODUZIR CÓDIGO FORA DA STDLIB.

- ▶ Constraints -> pacote constraints.
- ▶ Slices -> pacote x/exp/slices
  - ▶ Binary Search
- ▶ Maps -> pacote x/exp/maps
- ▶ Equal, EqualFunc, Keys, Values, Clear, DeleteFunc, Clone

# DELETE BY INDEX

```
s := []int{1, 2, 3, 4}  
s = append(s[:1], s[3:]...)
```

vs.

```
s = slices.Delete(s, 1, 3)
```

Powered by stdlib.

# POR QUE NÃO NO FUTURO?

**PARAMETRIZAVEL**

sync.Pool[T]  
sync.Map[K, V]  
atomic.Value[T]  
list.List[T]  
math.Abs[T]  
math.Min[T]  
math.Max[T]

**70 TRILHÕES DE VERSÕES**

sync.Pool  
sync.Map  
atomic.Value  
list.List  
math.Abs  
math.Min  
math.Max

**IDIOMATIC GO IS DEAD! LONG  
LIVE IDIOMATIC GO.**

**ATO FINAL  
E AGORA?**

**EU PRECISO MUDAR MINHA  
CODEBASE IMEDIATAMENTE?**

**EU PRECISO MUDAR MINHA CODEBASE  
IMEDIATAMENTE?  
NÃO!**

# QUANDO ALTERAR?

**QUANDO ALTERAR?  
SÓ ALTERE CÓDIGO COM UMA  
MOTIVAÇÃO.**

**QUANDO EU VER UMA  
INTERFACE {} EU  
PRECISO TROCAR POR ANY ?**

**QUANDO EU VER UMA INTERFACE {} EU  
PRECISO TROCAR POR ANY ?  
YES-ISH.**

**É SEGURO É UTILIZAR  
GENERICOS EM PRODUÇÃO?**

# É SEGURO É UTILIZAR GENERICS EM PRODUÇÃO?

Não! Nada é seguro de ser feito em produção. Faça sua gestão de risco normalmente.

**SEU PROGRAMA NÃO VAI  
PARAR DE RODAR POR USAR  
GENERICOS.**

**BACKWARD COMPATIBLE?**  
**YES-ISH.**

# O CÓDIGO SE Torna MAIS COMPLEXO?

**SIM!**

**Código abstrato é objetivamente mais complexo que código concreto.**

**SEMPRE QUE UM CÓDIGO SE  
REPETIR EU DEVO FAZER A  
VERSÃO GENÉRICA DELE?**

**BE AN ENGINEER NOT A  
PROGRAMMER.**

**OBRIGADO!**