



construção e uso

PRÁTICA DE CONJUNTOS

Porquê e como implementar um tipo Set na linguagem Go.

OBJETIVOS

1

Demonstrar como e porque o uso de conjuntos pode simplificar a lógica.

2

Discutir alternativas para implementar coleções de tipo Set em Go.

MOTIVAÇÃO

Quando senti saudade de conjuntos

CASO DE USO #1

EXIBIR PRODUTO SE
TODAS AS PALAVRAS
DA CONSULTA APARE-
CEREM NA DESCRIÇÃO

SOLUÇÃO SEM CONJUNTOS #1

Já escrevi código assim...

```
func ContainsAll(slice, subslice []string) bool {
    for _, needle := range subslice {
        found := false
        for _, elem := range slice {
            if needle == elem {
                found = true
                break
            }
        }
        if !found {
            return false
        }
    }
    return true
}
```

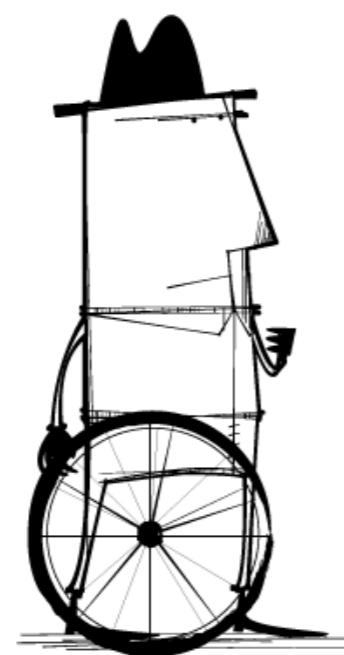
SOLUÇÃO SEM CONJUNTOS #2

Assim fica menos mal...

```
func Contains(slice []string, needle string) bool {
    for _, elem := range slice {
        if needle == elem {
            return true
        }
    }
    return false
}

func ContainsAll(slice, subslice []string) bool {
    for _, needle := range subslice {
        if !Contains(slice, needle) {
            return false
        }
    }
    return true
}
```

Que tal
se...



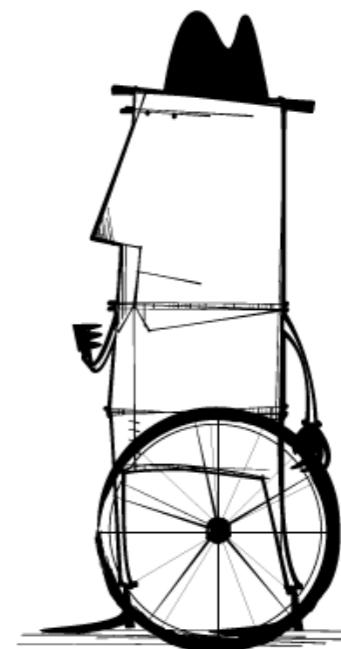
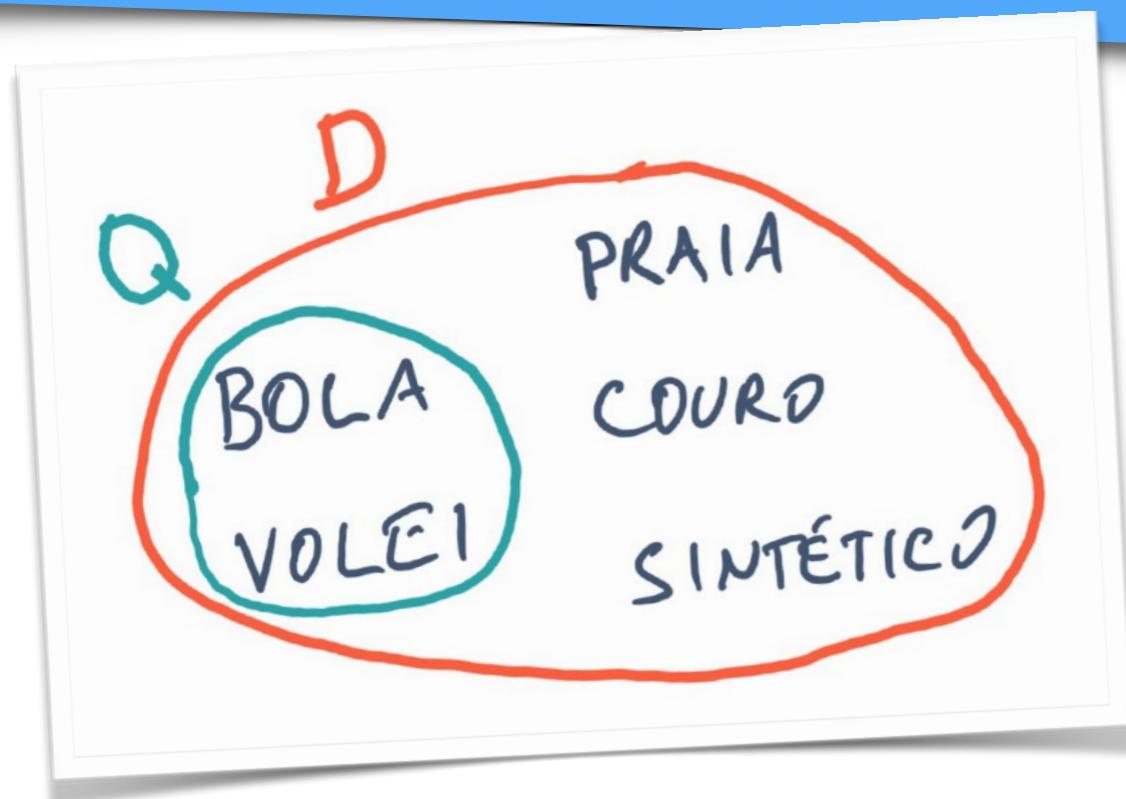
Depois! Agora estou
muito ocupado!



CASO DE USO #1

www.workcompass.com/

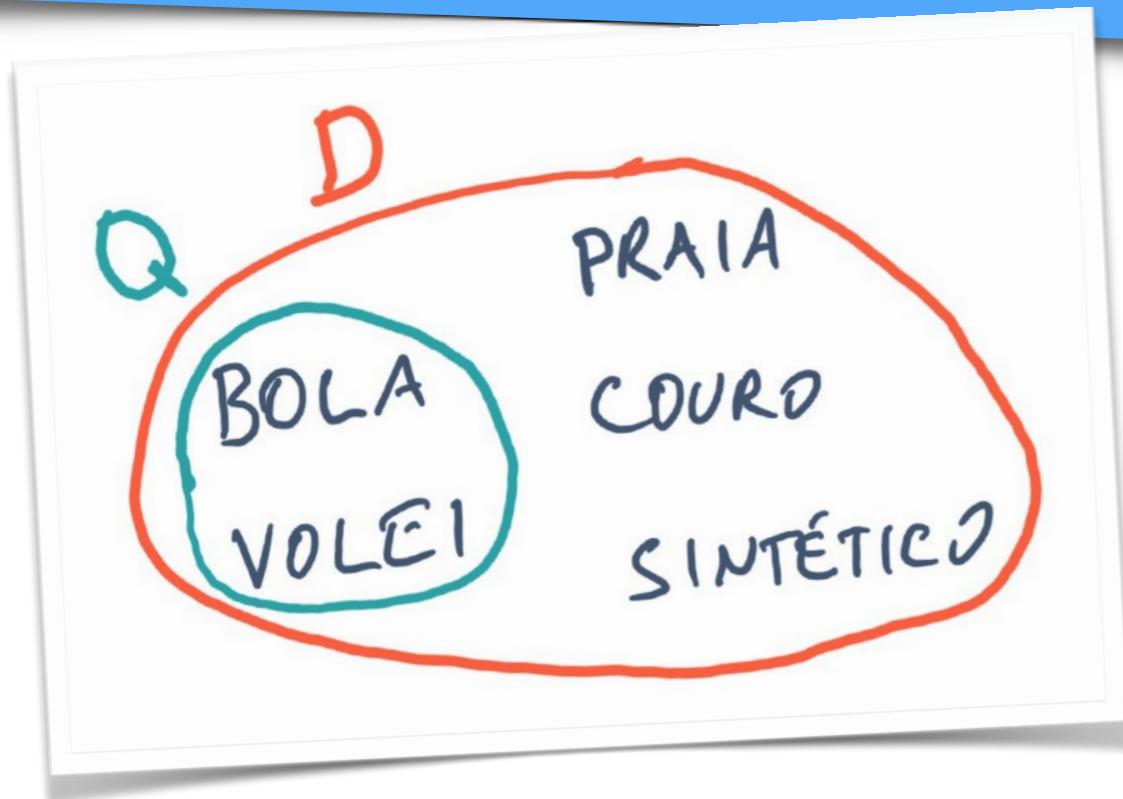
EXIBIR PRODUTO SE
TODAS AS PALAVRAS
DA CONSULTA APARE-
CEREM NA DESCRIÇÃO



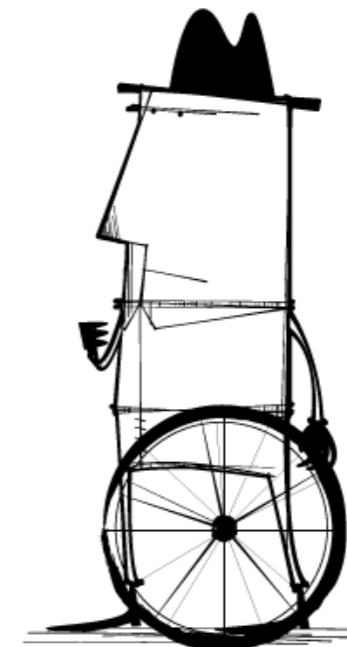
CASO DE USO #1

www.workcompass.com/

EXIBIR PRODUTO SE
TODAS AS PALAVRAS
DA CONSULTA APARE-
CEREM NA DESCRIÇÃO



$Q \subset D$

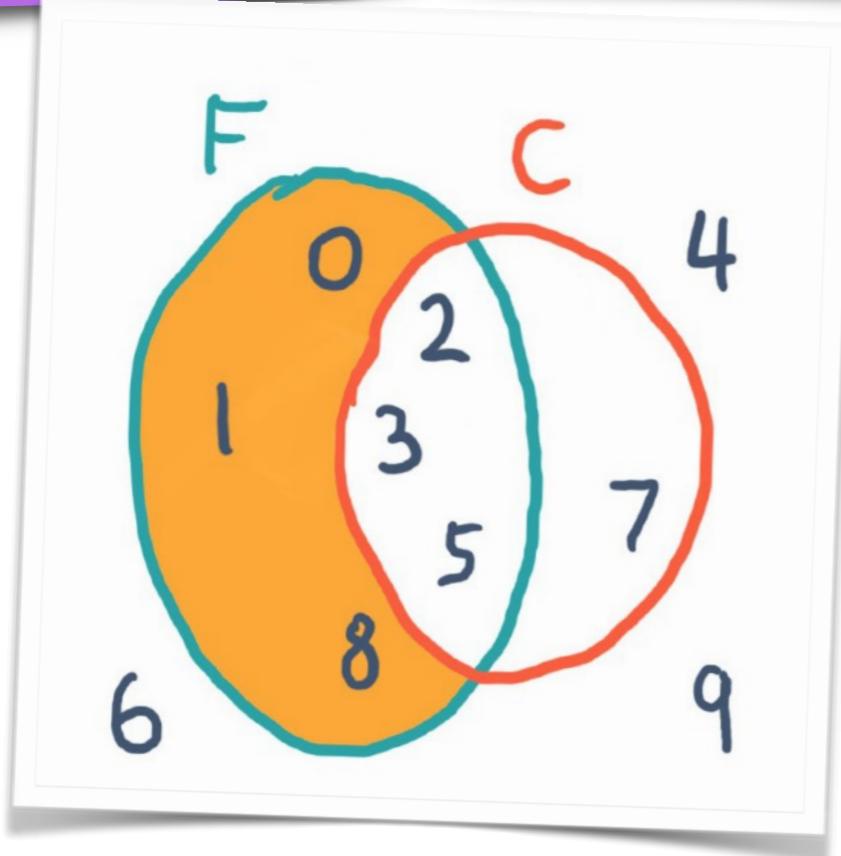


CASO DE USO #2

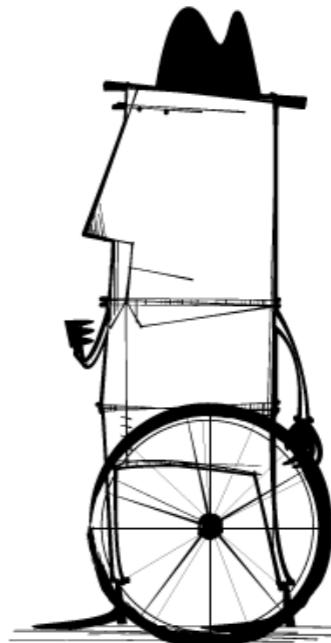
Marcar todos os produtos favoritados, exceto os que já estiverem no carrinho de compras.

CASO DE USO #2

Marcar todos os produtos favoritados, exceto os que já estiverem no carrinho de compras.



$$F \setminus C$$



CASO DE USO #3: RUNEFINDER

The screenshot shows a web browser window titled "Runefinder". The address bar displays the URL <https://runefinder2018.appspot.com/?q=face>. In the search bar, the word "face" is typed, and the "find" button is visible. Below the search bar, there is a link "Examples: bismillah box cat chess circle led". A "Fork me on GitHub" button is located in the top right corner of the page.

162 characters found

U+2639	⌚ WHITE FROWNING FACE
U+263A	☺ WHITE SMILING FACE
U+263B	☻ BLACK SMILING FACE
U+2680	Ѡ DIE FACE-1
U+2681	Ѡ DIE FACE-2
U+2682	Ѡ DIE FACE-3
U+2683	Ѡ DIE FACE-4
U+2684	Ѡ DIE FACE-5
U+2685	Ѡ DIE FACE-6
U+2FAF	⾯ KANGXI RADICAL FACE
U+3020	〠 POSTAL MARK FACE
U+FACE	〠 CJK COMPATIBILITY IDEOGRAPH-FACE
U+1DA07	〠 SIGNWRITING FACE DIRECTION POSITION NOSE FORWARD TILTING

<https://tgo.li/runes2018>

CASO DE USO #3: RUNEFINDER

The screenshot shows two browser windows side-by-side, both displaying the Runefinder website at <https://runefinder2018.appspot.com/>. The left window has a search bar containing 'face'. The right window has a search bar containing 'cat'. Both windows show a list of characters found, with the right window's list starting with:

Code Point	Category	Character	Description
U+2639	○ WH	⌚	WHEN
U+263A	○ WH	⌚	WHEN
U+263B	● BL	⌚	YI SYLLABLE CAT
U+2680	□ DI	⌚	PHAISTOS DISC SIGN CAT
U+2681	□ DI	🐱	CAT
U+2682	□ DI	😺	CAT FACE
U+2683	□ DI	😸	GRINNING CAT FACE WITH SMILING EYES
U+2684	□ DI	😹	CAT FACE WITH TEARS OF JOY
U+2685	□ DI	😻	SMILING CAT FACE WITH OPEN MOUTH
U+2FAF	面 KA	😺	SMILING CAT FACE WITH HEART-SHAPED EYES
U+3020	-NLS PO	😺	CAT FACE WITH WRY SMILE
U+FACE	[FR] CJ	😺	KISSING CAT FACE WITH CLOSED EYES
U+1DA07	[NLD ROT] SI TI	😺	POUTING CAT FACE
		😿	CRYING CAT FACE
		🙀	WEARY CAT FACE

A 'Fork me on GitHub' button is visible in the bottom right corner of the right-hand window.

<https://tgo.li/runes2018>

CASO DE USO #3: RUNEFINDER

The screenshot shows three separate browser windows, each titled "Runefinder", demonstrating the use of the service. The tabs are open to different URLs:

- The first tab has the URL <https://runefinder2018.appspot.com/?q=face> and displays a list of characters related to faces.
- The second tab has the URL <https://runefinder2018.appspot.com/?q=cat> and displays a list of characters related to cats.
- The third tab has the URL <https://runefinder2018.appspot.com/?q=face+cat> and displays a list of characters found by combining the previous two queries.

The results for the combined query are as follows:

Character	Code Point	Description
🐱	U+1F431	CAT FACE
😺	U+1F638	GRINNING CAT FACE WITH SMILING EYES
😸	U+1F639	CAT FACE WITH TEARS OF JOY
😹	U+1F63A	SMILING CAT FACE WITH OPEN MOUTH
😻	U+1F63B	SMILING CAT FACE WITH HEART-SHAPED EYES
😾	U+1F63C	CAT FACE WITH WRY SMILE
😽	U+1F63D	KISSING CAT FACE WITH CLOSED EYES
😿	U+1F63E	POUTING CAT FACE
😾	U+1F63F	CRYING CAT FACE
😾	U+1F640	WEARY CAT FACE

A "Fork me on GitHub" button is visible in the bottom right corner of the third tab's interface.

<https://tgo.li/runes2018>

CASO DE USO #3: ÍNDICE INVERTIDO

```
func Example_invertedIndex() {
    index := make(map[string]Set)
    index["SIGN"] = Make('#', '$', '%', '+', '<', '§', '©', '¬', ®, °, ±, µ, ¶,
    index["REGISTERED"] = Make(®)
    index["CHESS"] = Make(♣, ♚, ♗, ♔, ♕, ♛, ♙, ♖, ♘, ♜, ♞, ♟, ♢, ♡)
    index["BLACK"] = Make(⚑, ♜, ♞, ♟, ♢, ♡, ♖, ♘, ♜, ♞, ♟, ♢, ♡)
    result := index["CHESS"].Intersection(index["BLACK"])
    fmt.Println(result)
    // Output:
    // Set{ ♘ ♜ ♞ ♟ ♢ ♡ }
```

CASO DE USO #3: ÍNDICE INVERTIDO

```
func Example_invertedIndex() {  
    index := make(map[string]Set)  
    index["SIGN"] = Make('#', '$', '%', '+', '<', '§', '©', '¬', '®', '°', '±', 'μ', '¶',  
    index["REGISTERED"] = Make('®')  
    index["CHESS"] = Make('♙', '♘', '♗', '♕', '♔', '♚', '♝', '♞', '♜', '♝', '♞', '♜', '♟', '♞', '♝', '♞', '♝', '♟',  
    index["BLACK"] = Make('♟', '♞', '♝', '♛', '♚', '♝', '♞', '♟', '♝', '♞', '♝', '♞', '♝', '♟', '♞', '♝', '♞', '♝', '♟',  
    result := index["CHESS"].Intersection(index["BLACK"])  
    fmt.Println(result)  
    // Output:  
    // Set{♟ ♞ ♛ ♕ ♚ ♝ ♞ ♜ ♟ ♝ ♞ ♜ ♟ ♙ ♞ ♝ ♞ ♜ ♟ ♙}
```

"REGISTERED SIGN" é o nome oficial de ® em UnicodeData.txt

CASO DE USO #3: ÍNDICE INVERTIDO

```
func Example_invertedIndex() {  
    index := make(map[string]Set)  
    index["SIGN"] = Make('#', '$', '%', '+', '<', '§', '©', '¬', '®', '°', '±', 'µ', '¶'  
    index["REGISTERED"] = Make('®')  
    index["CHESS"] = Make('♙', '♘', '♗', '♕', '♔', '♚', '♝', '♞', '♜', '♝', '♞', '♜')  
    index["BLACK"] = Make('♟', '♞', '♝', '♛', '♚', '♝', '♞', '♟', '♝', '♞', '♜', '♝', '♞', '♜')  
    result := index["CHESS"].Intersection(index["BLACK"])  
    fmt.Println(result)  
    // Output:  
    // Set{♙ ♘ ♗ ♕ ♔ ♚ ♛ ♞ ♜ ♝ ♞ ♜}  
}
```

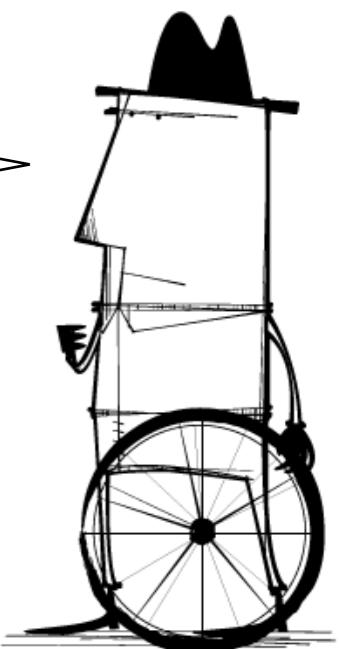
index é o índice invertido:
as chaves são palavras, e
os valores são conjuntos
de runas cujo nome
contém a palavra. Ex. a
runa '®' aparece nas
chaves "REGISTERED" e
"SIGN"

CASO DE USO #3: ÍNDICE INVERTIDO

```
func Example_invertedIndex() {
    index := make(map[string]Set)
    index["SIGN"] = Make('#', '$', '%', '+', '<', '§', '©', '¬', '®', '°', '±', 'µ', '¶',
    index["REGISTERED"] = Make('®')
    index["CHESS"] = Make('♙', '♘', '♗', '♕', '♔', '♚', '♝', '♛', '♜', '♝', '♞', '♟')
    index["BLACK"] = Make('♟', '♞', '♝', '♛', '♚', '♝', '♞', '♟', '♝', '♛', '♚', '♝', '♞', '♟')
    result := index["CHESS"].Intersection(index["BLACK"])
    fmt.Println(result)
    // Output:
    // Set{♙ ♘ ♗ ♛ ♚ ♗ ♞ ♟ ♗ ♛ ♚ ♗ ♞ ♟}
}
```



$$A \cap B$$



LÓGICA E CONJUNTOS

Uma relação íntima

Nobody has yet discovered a branch of mathematics that has successfully resisted formalization into set theory.

*Thomas Forster
Logic Induction and Sets, p. 167*

Ainda não descobriram um ramo da matemática que não possa ser formalizado pela teoria dos conjuntos.

Thomas Forster

(tradução livre de Logic Induction and Sets, p. 167)

CONJUNÇÃO LÓGICA: INTERSECÇÃO

x pertence à intersecção de A e B .

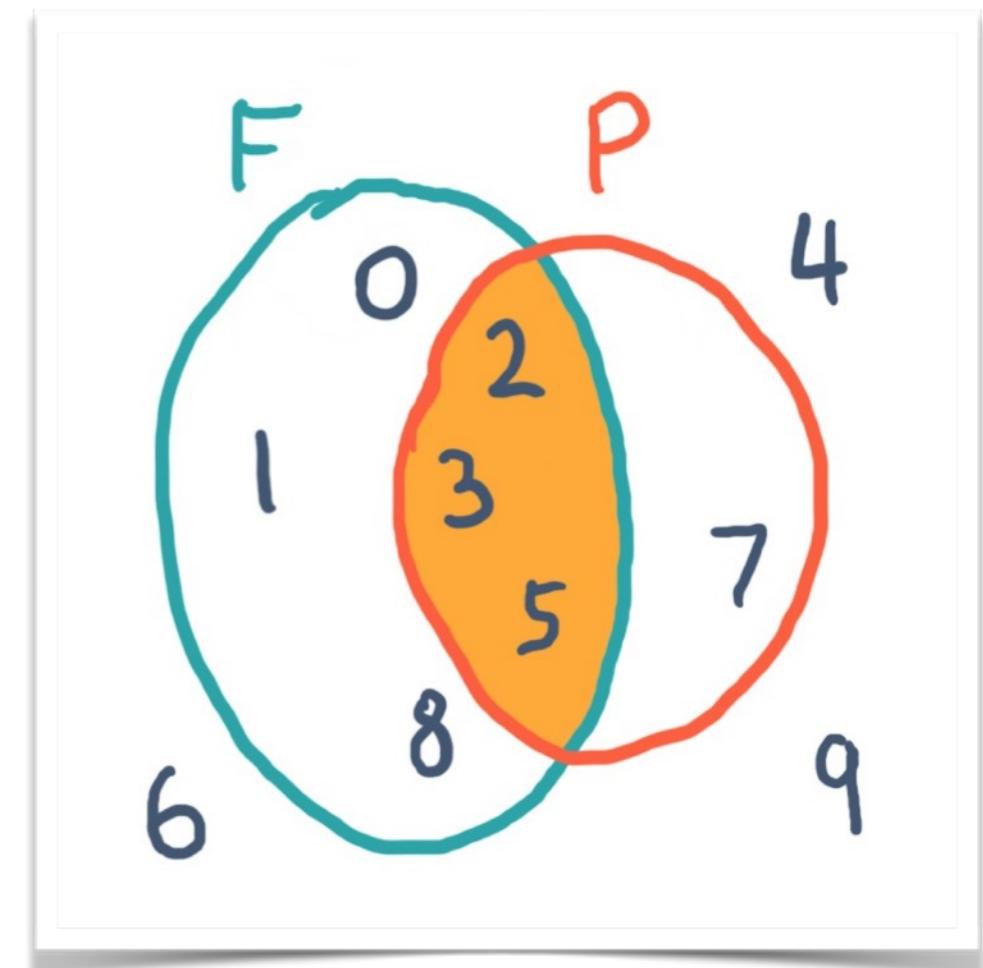
é o mesmo que:

x pertence ao conjunto A e

x também pertence ao conjunto B .

Em notação matemática:

$$x \in (A \cap B) \iff (x \in A) \wedge (x \in B)$$



Em computação: **AND**

DISJUNÇÃO LÓGICA: UNIÃO

x pertence à união de A e B .

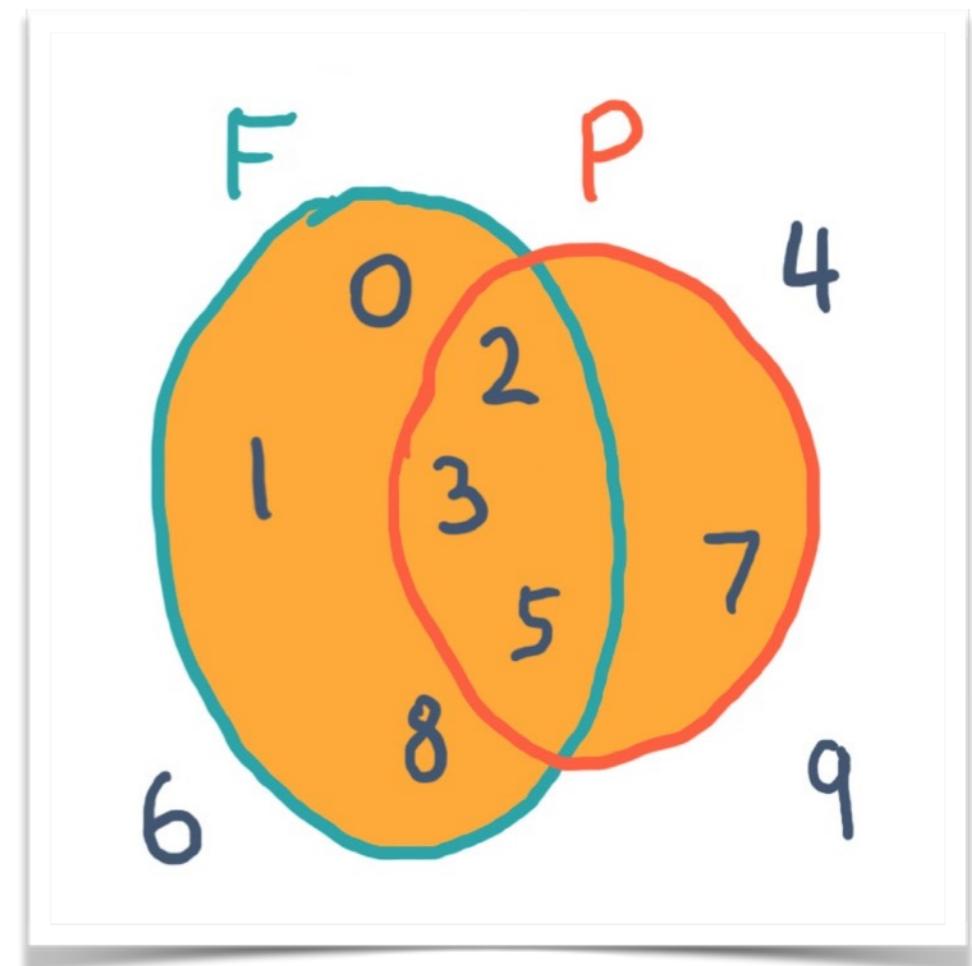
é o mesmo que:

x pertence ao conjunto A e/ou

x pertence ao conjunto B .

Em notação matemática:

$$x \in (A \cup B) \iff (x \in A) \vee (x \in B)$$



Em computação: **OR**

DIFERENÇA SIMÉTRICA

x pertence ao conjunto A ou

x pertence ao conjunto B

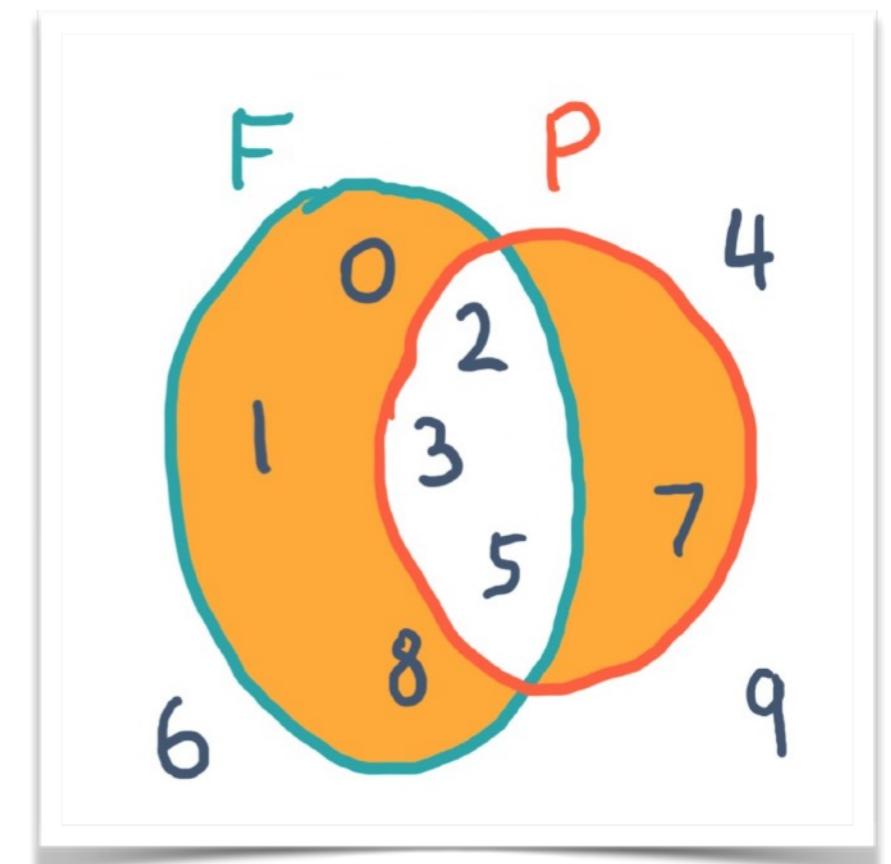
mas não pertence aos dois

é o mesmo que:

x pertence à união de A e B menos a intersecção de A e B .

Em notação matemática:

$$x \in (A \Delta B) \iff (x \in A) \vee (x \in B)$$



Em computação: **XOR**

DIFERENÇA

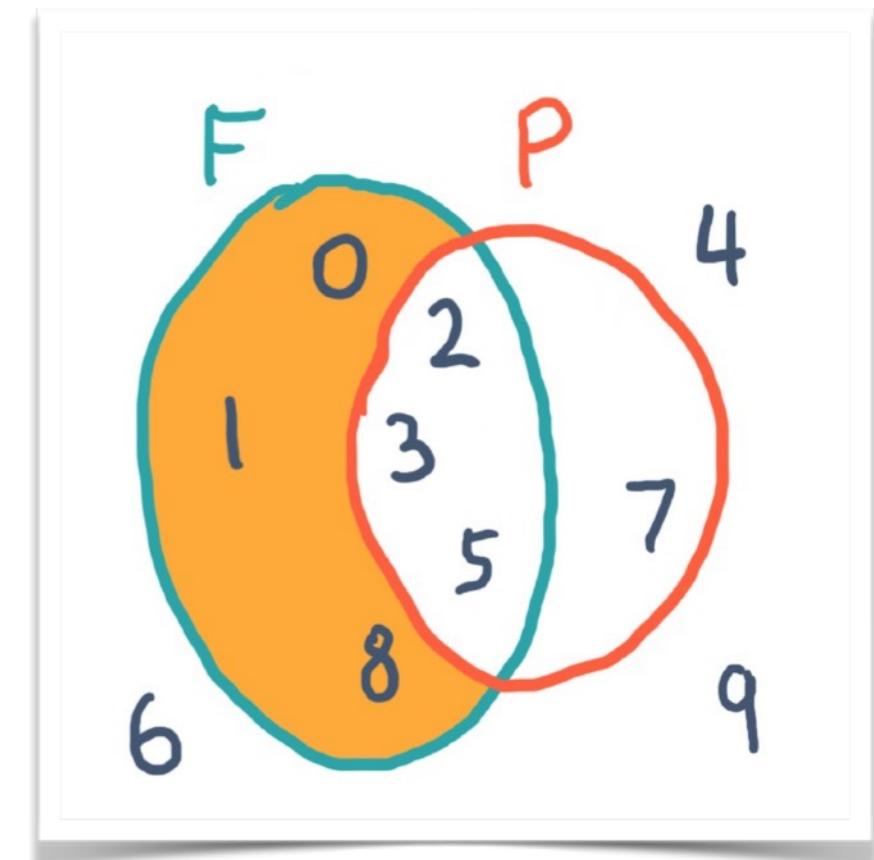
*x pertence ao conjunto A mas
não pertence ao conjunto B.*

é o mesmo que:

elementos de A menos os de B

Em notação matemática:

$$x \in (A \setminus B) \iff (x \in A) \wedge (x \notin B)$$



CONJUNTOS EM VÁRIAS LINGUAGENS

CONJUNTOS EM VÁRIAS BIBLIOTECAS-PADRÃO

Algumas linguagens/plataformas que implementam conjuntos em sua biblioteca-padrão.

Java Interface **Set** com < 10 métodos; 8 implementações

Python **set** e **frozenset** com > 10 métodos e operadores

.Net (C# etc.) Interface **ISet** com > 10 métodos; 2 implementações

JavaScript (ES6) **Set** com < 10 métodos

Ruby **Set** com > 10 métodos e operadores

CONJUNTOS “GENÉRICOS” EM GO

Elementos tipo `interface{}`

ThoughtWorks®

GOLANG-SET

By Ralph Caraveo (@deckarep)

deckarep/golang-set: A simple set type for the Go language. Also used in Docker.

This repository Search Pull requests Issues Marketplace Explore

deckarep / golang-set Watch 22 Star 718 Fork 82

Code Issues 4 Pull requests 0 Projects 0 Insights

A simple set type for the Go language. Also used in Docker.

go set threadsafe datastructures

114 commits 7 branches 4 releases 18 contributors

Branch: master New pull request Create new file Upload files Find file Clone or download

lorneli and deckarep threadUnsafeSet: reduce allocation in Add func ... Latest commit b87793d 23 days ago

File	Commit Message	Date
.gitignore	Initial commit	5 years ago
.travis.yml	Update Travis for 1.9	8 months ago
LICENSE	Adding MIT license headers and LICENSE file	5 years ago
README.md	Added a Go Report Card badge	11 months ago
bench_test.go	add Each method	6 months ago
iterator.go	Casual refactor (#43)	a year ago
iterator_example_test.go	Addressed issues reported by golint	11 months ago
set.go	add Each method	6 months ago

GOLANG-SET, VULGO MAPSET...

The screenshot shows a web browser window displaying the GoDoc documentation for the `mapset` package from GitHub. The URL in the address bar is `https://godoc.org/github.com/deckarep/golang-set`. The page title is "mapset - GoDoc". The main content area shows the package name `golang-set` and its GitHub repository. Below this, the title "package mapset" is displayed, followed by the import statement `import "github.com/deckarep/golang-set"`. A detailed description follows, stating that the package implements a simple and generic set collection with unique items and supports various set operations like membership testing, intersection, union, difference, symmetric difference, and cloning. Another paragraph describes the package's two implementations of the Set interface, noting the default safe concurrent access and the non-thread-safe implementation for performance benefits. The bottom section, titled "Index", lists the types defined in the package: `Iterator`, `OrderedPair`, and `Set`, each with a corresponding list of methods or functions.

golang-set: github.com/deckarep/golang-set [Index](#) | [Examples](#) | [Files](#)

package mapset

```
import "github.com/deckarep/golang-set"
```

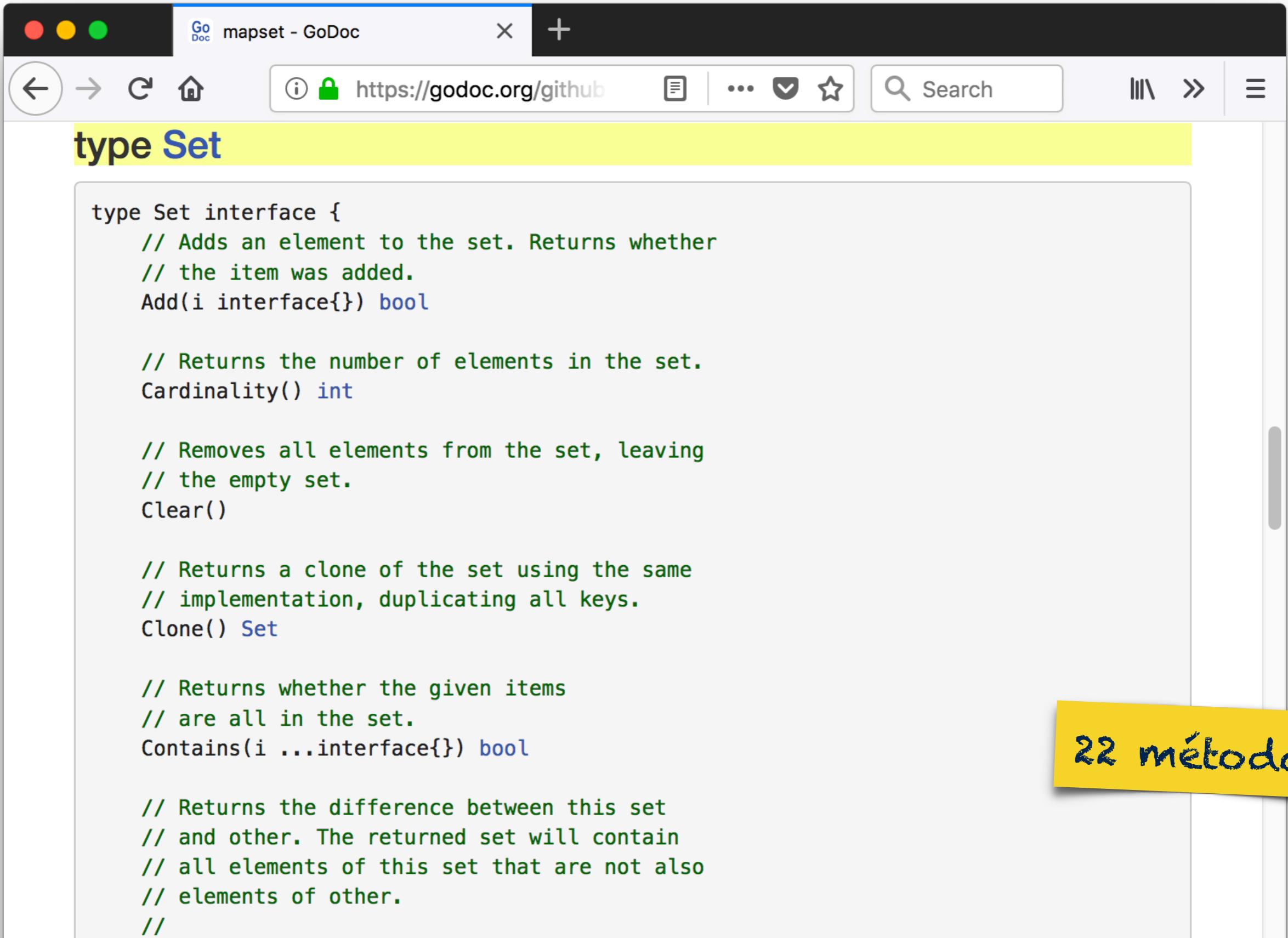
Package mapset implements a simple and generic set collection. Items stored within it are unordered and unique. It supports typical set operations: membership testing, intersection, union, difference, symmetric difference and cloning.

Package mapset provides two implementations of the Set interface. The default implementation is safe for concurrent access, but a non-thread-safe implementation is also provided for programs that can benefit from the slight speed improvement and that can enforce mutual exclusion through other means.

Index

- type `Iterator`
 - o func (*i* *`Iterator`) `Stop()`
- type `OrderedPair`
 - o func (*pair* *`OrderedPair`) `Equal(other OrderedPair) bool`
 - o func (*pair* `OrderedPair`) `String() string`
- type `Set`
 - o func `NewSet(s ...interface{}) Set`
 - o func `NewSetFromSlice(s []interface{}) Set`
 - o func `NewSetWith(elts ...interface{}) Set`
 - o func `NewThreadSafeSet() Set`

GOLANG-SET: INTERFACE SET



The screenshot shows a browser window with the title "mapset - GoDoc". The address bar contains the URL "https://godoc.org/github.com/golang/set". The main content area displays the Go code for the `Set` interface. A yellow bar at the top highlights the word "type". A yellow callout box in the bottom right corner contains the text "22 métodos".

```
type Set interface {
    // Adds an element to the set. Returns whether
    // the item was added.
    Add(i interface{}) bool

    // Returns the number of elements in the set.
    Cardinality() int

    // Removes all elements from the set, leaving
    // the empty set.
    Clear()

    // Returns a clone of the set using the same
    // implementation, duplicating all keys.
    Clone() Set

    // Returns whether the given items
    // are all in the set.
    Contains(i ...interface{}) bool

    // Returns the difference between this set
    // and other. The returned set will contain
    // all elements of this set that are not also
    // elements of other.
    //
```

22 métodos

GOLANG-SET: IMPLEMENTAÇÕES

threadUnsafeSet

```
type threadUnsafeSet map[interface{}]struct{}

func newThreadUnsafeSet() threadUnsafeSet {
    return make(threadUnsafeSet)
}
```

threadSafeSet

```
type threadSafeSet struct {
    s threadUnsafeSet
    sync.RWMutex
}

func newThreadSafeSet() threadSafeSet {
    return threadSafeSet{s: newThreadUnsafeSet()}
}
```

GOLANG-SET: CODE REVIEW

elementos tipo interface{}

```
func (set *threadUnsafeSet) Add(i interface{}) bool {
    _, found := (*set)[i]
    if found {
        return false //False if it existed already
    }

    (*set)[i] = struct{}{}
    return true
}

func (set *threadUnsafeSet) Contains(i ...interface{}) bool {
    for _, val := range i {
        if _, ok := (*set)[val]; !ok {
            return false
        }
    }
    return true
}
```

GOLANG-SET: EXEMPLO

```
func Example_invertedIndex() {
    index := make(map[string]mapset.Set)
    index["CHESS"] = mapset.NewSet('♔', '♕', '♗', '♘', '♙', '♚', '♛', '♝', '♞', '♟')
    index["BLACK"] = mapset.NewSet('♝', '♞', '♟', '★', '♚', '♛', '♝', '♞', '♟')
    result := index["CHESS"].Intersect(index["BLACK"])
    // fmt.Println(result) outputs in random order:
    // Set{9818, 9820, 9821, 9823, 9819, 9822}
    // The next five lines make the result testable:
    list := []string{}
    for char := range result.Iter() {
        list = append(list, string(char.(rune)))
    }
    sort.Strings(list)
    fmt.Println(list)
    // Output:
    // [♔ ♕ ♗ ♘ ♙ ♚ ♛ ♝ ♞ ♟]
}
```



GOLANG-SET: MORE CODE REVIEW

GOLANG-SET: CODE REVIEW

```
func (set *threadUnsafeSet) Difference(other Set) Set {
    _ = other.(*threadUnsafeSet)

    difference := newThreadUnsafeSet()
    for elem := range *set {
        if !other.Contains(elem) {
            difference.Add(elem)
        }
    }
    return &difference
}
```

GOLANG-SET: CODE REVIEW

```
func (set *threadUnsafeSet) IsSubset(other Set) bool {
    _ = other.(*threadUnsafeSet)
    for elem := range *set {
        if !other.Contains(elem) {
            return false
        }
    }
    return true
}

func (set *threadUnsafeSet) IsProperSubset(other Set) bool {
    return set.IsSubset(other) && !set.Equal(other)
}
```

ThoughtWorks®

GODS

Go Data Structures

elementos tipo interface{}

The screenshot shows a GitHub repository page for 'emirasic/gods'. The repository title is 'emirasic / gods'. It has 188 stars and 383 forks. The repository description is: 'GoDS (Go Data Structures). Containers (Sets, Lists, Stacks, Maps, Trees), Sets (HashSet, TreeSet), Lists (ArrayList, SinglyLinkedList, DoublyLinkedList), Stacks (LinkedListStack, ArrayStack), Maps (HashMap, TreeMap, HashBidiMap, TreeBidiMap), Trees (RedBlackTree, AVLTree, BTree, BinaryHeap), Comparators, Iterators, Enumerables, Sort, JSON'. The repository has 275 commits, 1 branch, 13 releases, and 11 contributors. The commit history includes several updates to containers, examples, lists, maps, and sets, with the latest commit being 'b2394df' on Dec 26, 2017.

emirasic/gods: GoDS (Go Data Structures)

This repository Search

Pull requests Issues Marketplace Explore

Watch 188 Star 4,025 Fork 383

Code Issues 11 Pull requests 2 Projects 0 Wiki Insights

GoDS (Go Data Structures). Containers (Sets, Lists, Stacks, Maps, Trees), Sets (HashSet, TreeSet), Lists (ArrayList, SinglyLinkedList, DoublyLinkedList), Stacks (LinkedListStack, ArrayStack), Maps (HashMap, TreeMap, HashBidiMap, TreeBidiMap), Trees (RedBlackTree, AVLTree, BTree, BinaryHeap), Comparators, Iterators, Enumerables, Sort, JSON

go golang data-structure map tree set list stack iterator enumerable sort avl-tree red-black-tree b-tree binary-heap

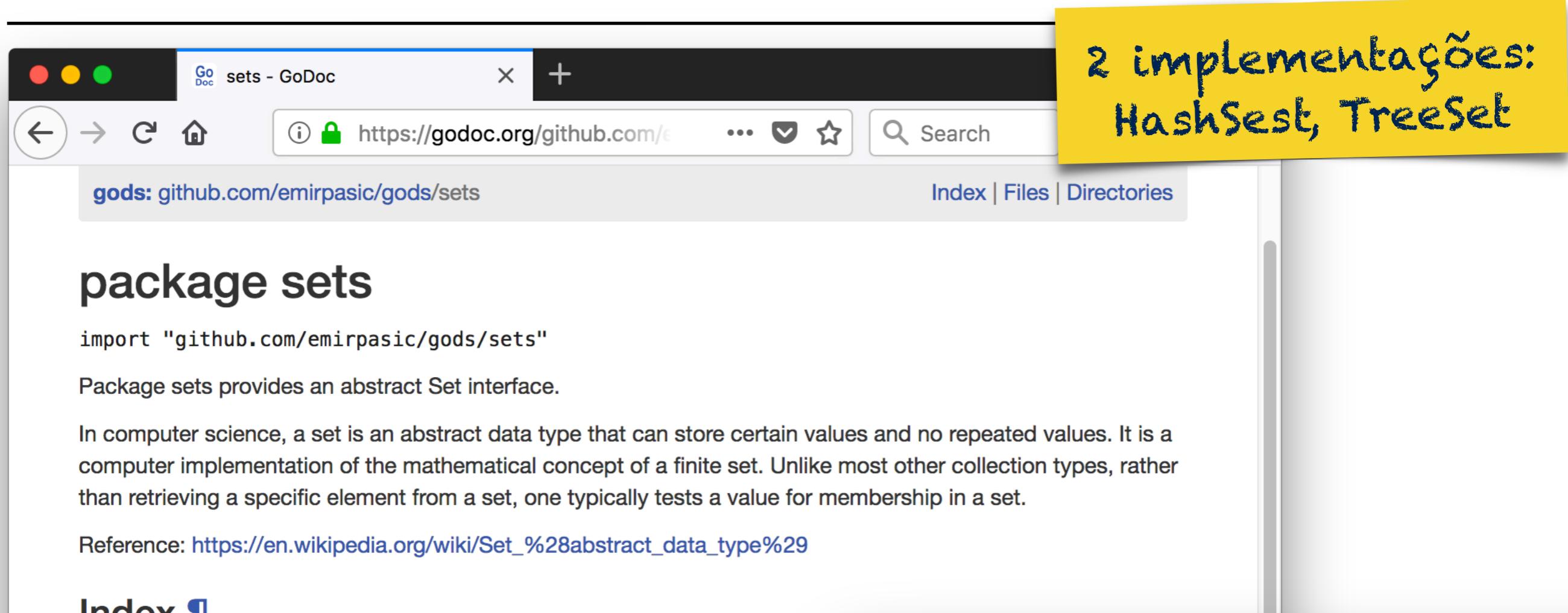
275 commits 1 branch 13 releases 11 contributors

Branch: master New pull request Create new file Upload files Find file Clone or download

emirasic committed on Dec 26, 2017 Merge pull request #69 from mahadevTW/indexOfMethod ... Latest commit b2394df on Dec 26, 2017

Commit	Message	Date
containers	Update enumerable.go	7 months ago
examples	- serialization example	a year ago
lists	Add IndexOf method to DoublyLinkedList	4 months ago
maps	- tree-map and tree-bidi-map (de)serialization	a year ago
sets	- JSON serialization for all sets	a year ago

GODS: PACKAGE SETS



2 implementações:
HashSet, TreeSet

godsets: github.com/emirpasic/gods/sets Index | Files | Directories

package sets

```
import "github.com/emirpasic/gods/sets"
```

Package sets provides an abstract Set interface.

In computer science, a set is an abstract data type that can store certain values and no repeated values. It is a computer implementation of the mathematical concept of a finite set. Unlike most other collection types, rather than retrieving a specific element from a set, one typically tests a value for membership in a set.

Reference: https://en.wikipedia.org/wiki/Set_%28abstract_data_type%29

Index ↗

[type Set](#)

[Package Files](#)

[sets.go](#)

type Set

```
type Set interface {
    Add(elements ...interface{})
    Remove(elements ...interface{})
    Contains(elements ...interface{}) bool
}
```

[containers.Container](#)

interface:
7 métodos



type Container

```
type Container interface {
    Empty() bool
    Size() int
    Clear()
    Values() []interface{}
}
```

Container is base interface that all data structures implement.

ThoughtWorks®

GEN

Type-driven code generation for Go

GEN: GERADOR DE CÓDIGO COM TIPOS PARAMETRIZADOS

The screenshot shows a web browser window with the URL clipperhouse.github.io/gen/. The page title is "gen (v4)" and the subtitle is "Type-driven code generation for Go". The left sidebar contains navigation links for "home", "Quick start", "Annotations", "Usage", "FAQ", "typewriters", "Listing", "Adding", and "Implementing". The main content area starts with a brief introduction about gen's purpose: "gen is an attempt to bring some generics-like functionality to Go. It uses type annotations to add ‘of T’ functionality to your packages." It then explains how gen generates code for types at development time, using the command line, and becomes part of the package with no external dependencies. The "Quick start" section provides instructions: "Of course, start by installing Go, [setting up paths](#), etc. Then:" followed by a code block:

```
go get github.com/clipperhouse/gen
```

. It continues with "Create a new Go project, and `cd` into it. Create a `main.go` file and define a type." and "Now, mark it up with a `+gen` annotation in an adjacent comment like so:" followed by a code block:

```
// +gen slice:"Where,Count,GroupBy[string]"  
type MyType struct {}
```

. Finally, it says "And at the command line, simply type:" followed by a code block:

```
gen
```

.

gen · type-driven code generati × +

home Quick start Annotations Usage FAQ typewriters Listing Adding Implementing slice Aggregate[T] All Any Average Average[T] Count Distinct DistinctBy First GroupBy[T] Max Max[T] MaxBy Min Min[T] MinBy

clipperhouse.github.io/gen/ Search source · changelog

gen (v4)

Type-driven code generation for Go

gen is an attempt to bring some generics-like functionality to Go. It uses type annotations to add “of T” functionality to your packages.

gen generates code for your types, at development time, using the command line. It is not an import; the generated source becomes part of your package and takes no external dependencies.

Quick start

Of course, start by installing Go, [setting up paths](#), etc. Then:

```
go get github.com/clipperhouse/gen
```

Create a new Go project, and `cd` into it. Create a `main.go` file and define a type.

Now, mark it up with a `+gen` annotation in an adjacent comment like so:

```
// +gen slice:"Where,Count,GroupBy[string]"  
type MyType struct {}
```

And at the command line, simply type:

```
gen
```

GEN: MODO DE USAR

1. Marque o seu código com comentários especiais:

```
// +gen set  
type Codepoint rune  
  
func (c Codepoint) String() string {
```

2. Execute o comando gen

3. Adicione ao seu projeto o arquivo .go gerado

GEN: MEU CÓDIGO FONTE (CODEPOINTS.GO)

```
// Package cptset implements a set of Codepoints (a.k.a. runes)
package cptset

import (
    "bytes"
    "sort"
)

// +gen set
type Codepoint rune

func (c Codepoint) String() string {
    return string(c)
}

// ToStringSlice returns elements as a slice of strings
func (set CodepointSet) ToStringSlice() []string {
    var s []string
    for v := range set {
        s = append(s, string(v))
    }
}
```

Exemplo em: <https://tgo.li/2vtcMjw> 5

GEN: O ARQUIVO GERADO (CODEPOINT_SET.GO)

```
// Generated by: main  
// TypeWriter: set  
// Directive: +gen on Codepoint
```

```
package cptset
```

```
// Set is a modification of https://github.com/deckarep/golang-set  
// The MIT License (MIT)  
// Copyright (c) 2013 Ralph Caraveo (deckarep@gmail.com)
```

```
// CodepointSet is the primary type that represents a set  
type CodepointSet map[Codepoint]struct{}
```

```
// NewCodepointSet creates and returns a reference to an empty set.  
func NewCodepointSet(a ...Codepoint) CodepointSet {  
    s := make(CodepointSet)  
    for _, i := range a {  
        s.Add(i)  
    }
```

código derivado
de golang-set

Exemplo em: <https://tgo.li/2vtcMjw> 5

CONJUNTOS ESPECIALIZADOS

ThoughtWorks®

MAP CRU

USO DE MAP COMO CONJUNTO

Chaves de um **map** formam um conjunto com duas características essenciais:

- Garantia de unicidade:
uma chave só pode ocorrer uma vez.
- Verificação de pertencimento em $O(1)$:
tempo praticamente constante independente do número de elementos.

```
if _, existe := conjunto[elem]; existe {  
    // o elemento está no conjunto  
}
```

Demais métodos? Fique à vontade para implementar!

USO DE MAP COMO CONJUNTO

Chaves de um **map** formam um conjunto com duas características essenciais:

- Garantia de unicidade:
uma chave só pode ocorrer uma vez.
- Verificação de pertencimento em $O(1)$:
tempo praticamente constante independente do número de elementos.

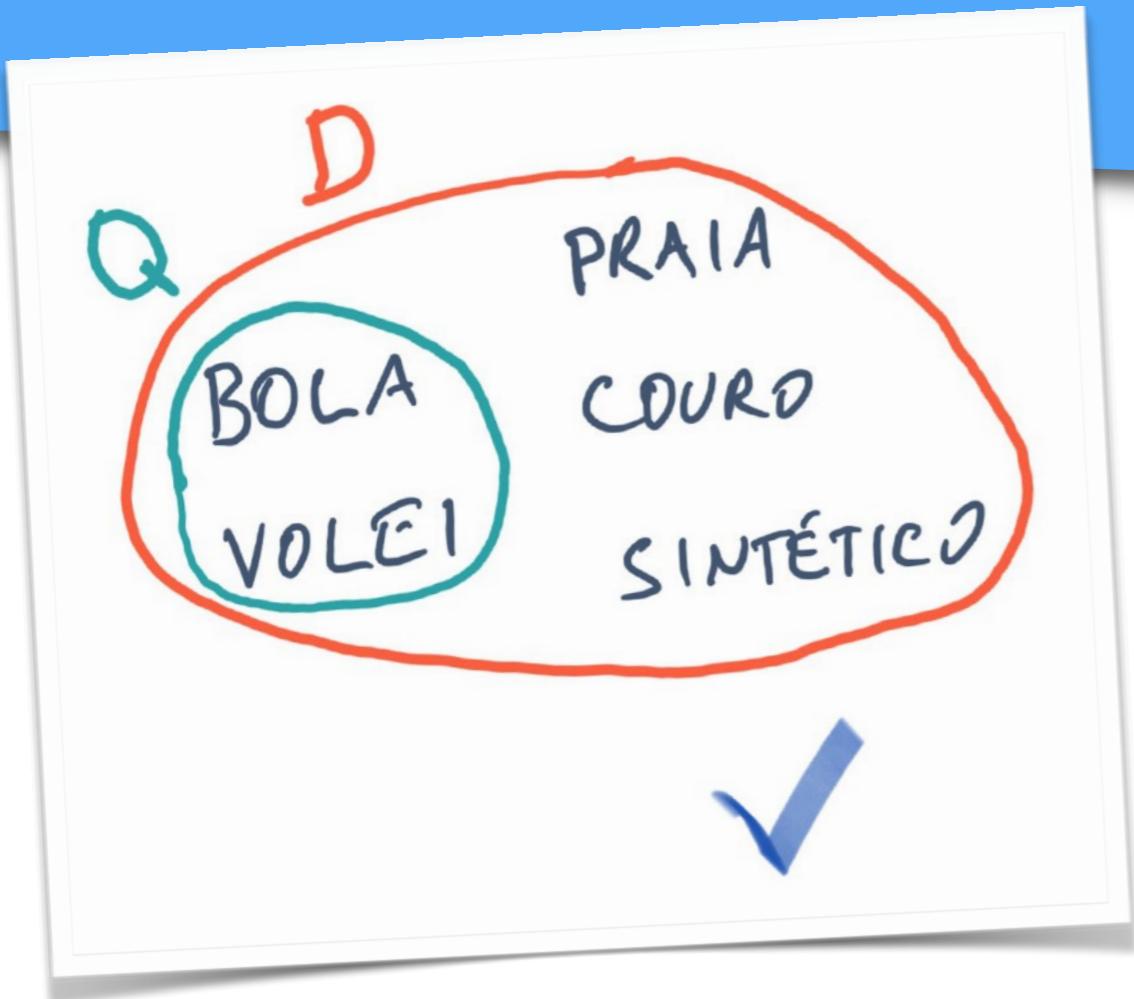
Curiosidade:
era assim que a gente
se virava em Python
até 2003 – usando
chaves de um dict
como um conjunto!

```
if _, existe := conjunto[element]; existe {  
    // o elemento está no conjunto  
}
```

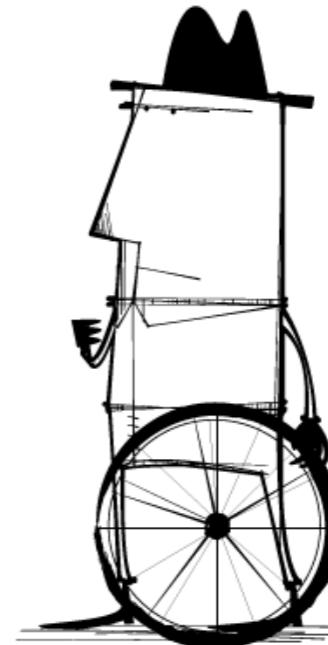
Demais métodos? Fique à vontade para implementar!

RELEMBRANDO: CASO DE USO #1

EXIBIR PRODUTO SE TODAS AS PALAVRAS DA CONSULTA APARECEREM NA DESCRIÇÃO



$$Q \subset D$$



SOLUÇÃO COM MAP FAZENDO PAPEL DE CONJUNTO

Solução mais eficiente que as anteriores, para slices grandes:

```
func ContainsAll(slice, subslice []string) bool {  
    set := make(map[string]struct{})  
    for _, elem := range slice {  
        set[elem] = struct{}{}  
    }  
    for _, needle := range subslice {  
        if _, found := set[needle]; !found {  
            return false  
        }  
    }  
    return true  
}
```

map de <tipo-elemento>
para struct{}

SOLUÇÃO COM MAP FAZENDO PAPEL DE CONJUNTO

Solução mais eficiente que as anteriores, para slices grandes:

```
func ContainsAll(slice, subslice []string) bool {
    set := make(map[string]struct{})
    for _, elem := range slice {
        set[elem] = struct{}{}
    }
    for _, needle := range subslice {
        if _, found := set[needle]; !found {
            return false
        }
    }
    return true
}
```

*preencher o
“conjunto”*

SOLUÇÃO COM MAP FAZENDO PAPEL DE CONJUNTO

Solução mais eficiente que as anteriores, para slices grandes:

```
func ContainsAll(slice, subslice []string) bool {
    set := make(map[string]struct{})
    for _, elem := range slice {
        set[elem] = struct{}{}
    }
    for _, needle := range subslice {
        if _, found := set[needle]; !found {
            return false
        }
    }
    return true
}
```

busca em $O(1)$

ThoughtWorks®

MAP COM NOME

standupdev/runeset

RUNESET: TIPO MAP COM NOME E MÉTODOS

The screenshot shows the GoDoc interface for the `runeset` package. The title bar says "runeset - GoDoc". The URL in the address bar is `https://godoc.org/github.com/standupdev/runeset`. Below the title, it says "package runeset". There is a code snippet: `import "github.com/standupdev/runeset"`. Two buttons are visible: "Example" and "Example (InvertedIndex)". A yellow callout bubble on the right side contains handwritten text: "10 métodos, incluindo: Intersection, IntersectionUpdate".

Index

type Set

- func `Make(chars ...rune) Set`
- func `MakeFromString(text string) Set`
- func `(s Set) Add(r rune)`
- func `(s Set) Contains(r rune) bool`
- func `(s Set) Copy() Set`
- func `(s Set) Equal(other Set) bool`
- func `(s Set) Intersection(other Set) Set`
- func `(s Set) IntersectionUpdate(other Set)`
- func `(s Set) Sorted() []rune`
- func `(s Set) String() string`

RUNESET: CARACERÍSTICAS

Não encapsula o map: pode acessado e modificado direto.

Sintaxe melhor para criar e testar elementos: `s.Add(e)`
`s.Contains(e)`

Exemplo: <https://github.com/standupdev/runeset>

RUNESET: DEFINIÇÃO DO TIPO E MÉTODOS ESSENCIAIS

```
// Set represents a set of runes
type Set map[rune]struct{}

// Add a rune to the set.
func (s Set) Add(r rune) {
    s[r] = struct{{}{}} // zero-byte struct
}

// Contains reports whether set contains given rune
func (s Set) Contains(r rune) bool {
    _, found := s[r]
    return found
}
```

elementos do tipo rune

RUNESET: CONSTRUTORES

Convenção:

Usar **Make** para criar objetos manipulados por referências.

```
// Make creates and returns a new Set
func Make(chars ...rune) Set {
    s := Set{}
    for _, c := range chars {
        s.Add(c)
    }
    return s
}

// MakeFromString creates and returns a new Set
func MakeFromString(text string) Set {
    return Make([]rune(text)...)
}
```

cria um
Set vazio



RUNESSET: CONSTRUTORES

MakeFromString: set de runas a partir de string

```
// Make creates and returns a new Set
func Make(chars ...rune) Set {
    s := Set{}
    for _, c := range chars {
        s.Add(c)
    }
    return s
}

// MakeFromString creates and returns a new Set
func MakeFromString(text string) Set {
    return Make([]rune(text)...)
}
```

custo zero
para passar
uma slice:
runas...

RUNESET: INTERSECÇÃO

```
// Intersection returns a new set: the intersection of s AND other
func (s Set) Intersection(other Set) Set {
    result := Set{}
    if len(other) > 0 {
        for r := range s {
            if other.Contains(r) {
                result.Add(r)
            }
        }
    }
    return result
}
```

// IntersectionUpdate changes receiver in-place, keeping only
// elements that are in it AND in other.

```
func (s Set) IntersectionUpdate(other Set) {
    for r := range s {
        if !other.Contains(r) {
            delete(s, r)
        }
    }
}
```

RUNESET: USO EM ÍNDICE INVERTIDO (CASO DE USO #3)

```
func Example_invertedIndex() {
    index := make(map[string]Set)
    index["SIGN"] = Make('#', '$', '%', '+', '<', '§', '©', '¬', ®, °, ±, µ, ¶,
    index["REGISTERED"] = Make(®)
    index["CHESS"] = Make(♦, ♞, ♜, ♚, ♗, ♕, ♖, ♛, ♞, ♜, ♚, ♗, ♕, ♖)
    index["BLACK"] = Make(¤, ■, ♚, ★, ♗, ♔, ♚, ♗, ♞, ♜, ♚, ♗, ♕, ♖)
    result := index["CHESS"].Intersection(index["BLACK"])
    fmt.Println(result)
    // Output:
    // Set{¤ ■ ♚ ★ ♗ ♔}
}
```

RUNESET: MÉTODOS ÚTEIS

Além de operações da teoria dos conjuntos...

MakeFromString: construtor que aceita string.

Sorted: devolve slice de runas em ordem ascendente

String: elementos ordenados para facilitar exemplos.

Equal: facilita muito os testes.

Copy: importante especialmente em coleções mutáveis.

ThoughtWorks®

MAP ENCAPSULADO

STRSET: TIPO MAP ENCAPSULADO EM STRUCT

The screenshot shows a browser window displaying the GoDoc documentation for the `strset` package. The title bar says "strset - GoDoc". The URL in the address bar is `https://godoc.org/github.com/standupdev/strset`. The main content area shows the package header:

package strset

```
import "github.com/standupdev/strset"
```

Package strset provides a Set type for string elements.

A blue button labeled "Example" is visible in the bottom-left corner of the content area.

Index

type Set

- func `Make(...string) Set`
- func `MakeFromText(text string) Set`
- func `(s Set) Add(elem string)`
- func `(s Set) AddAll(...string)`
- func `(s Set) Channel() <-chan string`
- func `(s *Set) Clear()`
- func `(s Set) Contains(elem string) bool`
- func `(s Set) ContainsAll(...string) bool`
- func `(s Set) Copy() Set`
- func `(s Set) Difference(other Set) Set`
- func `(s Set) DifferenceUpdate(other Set)`
- func `(s Set) Equal(other Set) bool`

A yellow callout bubble in the bottom-right corner contains the handwritten text "26 métodos".

STRSET: CARACTERÍSTICAS

Protege o map: campo privado só pode acessado via métodos e funções do mesmo pacote.

Não pode ser construído com **Set{}**: necessário usar **Make()**.

Exemplo: <https://github.com/standupdev/strset>

DEFINIÇÃO DO TIPO E CONSTRUTORES

```
// Set of strings.  
type Set struct {  
    store map[string]struct{}}  
  
// Make creates and returns a new Set.  
func Make(...string) Set {  
    s := Set{}  
    s.store = make(map[string]struct{})  
    for _, elem := range elems {  
        s.store[elem] = struct{}{}}  
    }  
    return s  
}  
  
// MakeFromText creates and returns a new Set from  
// a string of elements separated by whitespace.  
func MakeFromText(text string) Set {  
    return Make(strings.Fields(text)...)  
}
```

ORGANIZAÇÃO DO PACOTE STRSET

core.go

Make, Len, Contains, String, Equal, Copy,
ToSlice, Channel

operators.go

Intersection, Union,
Difference, SymmetricDifference

relations.go

SubsetOf, SupersetOf

updaters.go

Add, AddAll, Remove, RemoveAll, Pop, Clear,
IntersectionUpdate, UnionUpdate,
DifferenceUpdate, SymmetricDifferenceUpdate

Nota: todos os métodos que modificam um **Set** depois de criado
estão no arquivo **updaters.go**.

COMPARAR APIs PARA ESCOLHER NOMES DE MÉTODOS

API-comparison.ods

	A	B	C	D	E	F	G	H
1	Math symbol	Python operator	Python method	Java Set interface	gopl.io intset	emirpasic/ GoDS	deckarep/ golang-set	standupdev/ strset
2								
3	$S \cup Z$	$s z$	<code>s.union(it, ...)</code>				<code>s.Union(z)</code>	<code>s.Union(z)</code>
4		$s = z$	<code>s.update(it, ...)</code>		<code>s.UnionWith(z)</code>			<code>s.UnionUpdate(z)</code>
5				<code>s.addAll(c)</code>	<code>s.AddAll(e...)</code>	<code>s.Add(e...)</code>		<code>s.AddAll(e...)</code>
6			<code>s.add(e)</code>	<code>s.add(e)</code>	<code>s.Add(e)</code>	<code>s.Add(e...)</code>	<code>s.Add(e)</code>	<code>s.Add(e)</code>
7								
8	$S \cap Z$	$s & z$	<code>s.intersection(it, ...)</code>				<code>s.Intersection(z)</code>	<code>s.Intersection(z)</code>
9		$s &= z$	<code>s.intersection_update(it, ...)</code>	<code>s retainAll(c)</code>	<code>IntersectWith</code>			<code>s.IntersectionUpdate(z)</code>
10								
11	$S \setminus Z$	$s - z$	<code>s.difference(it, ...)</code>				<code>s.Difference(z)</code>	<code>s.Difference(z)</code>
12		$s -= z$	<code>s.difference_update(it, ...)</code>	<code>s.removeAll(c)</code>	<code>s.DifferenceWith(z)</code>			<code>s.DifferenceUpdate(z)</code>
13								<code>s.RemoveAll(e...)</code>
14	$S \Delta Z$	$s ^ z$	<code>s.symmetric_difference(it)</code>					<code>s.SymmetricDifference(z)</code>
15		$s ^= z$	<code>s.symmetric_difference_update</code>		<code>s.SymmetricDifference(z)</code>	<code>s.SymmetricDifference(z)</code>		<code>s.SymmetricDifferenceUpdate(z)</code>
16								
17	$e \in S$	$e \text{ in } s$	<code>s.__contains__(e)</code>	<code>s.contains(e)</code>	<code>s.Has(e)</code>			<code>s.Contains(e)</code>
18				<code>s.containsAll(c)</code>		<code>s.Contains(e)</code>	<code>s.Contains(e...)</code>	<code>s.ContainsAll(e...)</code>
19								
20	$S = Z$	$s == z$	<code>s.__eq__(z)</code>	<code>s.equals(o)</code>			<code>s.Equal(z)</code>	<code>s.Equal(z)</code>
21	$S \subseteq Z$	$s <= z$	<code>s.issubset(it)</code>				<code>s.IsSubset(z)</code>	<code>s.SubsetOf(z)</code>
22	$S \subset Z$	$s < z$	<code>s.__lt__(z)</code>				<code>s.IsProperSubset(z)</code>	
23	$S \supseteq Z$	$s >= z$	<code>s.issuperset(it)</code>					<code>s.SuperiorSubset(z)</code>
24	$S \supset Z$	$s > z$	<code>s.__gt__(z)</code>				<code>s.IsProperSuperset(z)</code>	
25								
26			<code>s.isdisjoint(z)</code>					
27	$S \times Z$						<code>s.CartesianProduct(z)</code>	
28							<code>s.PowerSet()</code>	
29								
30	<code>len(s)</code>	<code>s.__len__()</code>		<code>s.size()</code>	<code>s.Len()</code>	<code>s.Size()</code>	<code>s.Cardinality()</code>	<code>s.Len()</code>
31	<code>not s</code>	<code>s.__bool__()</code>				<code>s.Empty()</code>		
32		<code>s.__iter__()</code>		<code>s.toArray()</code>	<code>s.Elems()</code>	<code>s.Values()</code>	<code>s.ToSlice()</code>	<code>s.ToSlice()</code>
33								
34		<code>s.clear()</code>		<code>s.clear()</code>	<code>s.Clear()</code>	<code>s.Clear()</code>		<code>s.Clear()</code>
35		<code>s.copy()</code>			<code>s.Copy()</code>		<code>s.Clone()</code>	<code>s.Copy()</code>
36								

RUNESCAN: EXEMPLO DE USO DE STRSET

```
// ParseLine parses a line in the UnicodeData.txt file returning
// the rune, the name and a set of words build from the name.
func ParseLine(line string) (rune, string, strset.Set) {
    fields := strings.Split(line, ";")
    code, _ := strconv.ParseInt(fields[0], 16, 32)
    name := fields[1]
    wordStr := strings.Replace(fields[1], "-", " ", -1)
    words := strset.MakeFromText(wordStr)
    if fields[10] != "" {
        name += fmt.Sprintf(" (%s)", fields[10])
        wordStr = strings.Replace(fields[10], "-", " ", -1)
        words.AddAll(strings.Fields(wordStr)...)}
    return rune(code), name, words
}
```

```
// filter returns a list where each item is a [3]string with the
// U+XXXX codepoint, the character (as a string) and the name of the
// Unicode characters whose name contains all words in the query.
func filter(text io.Reader, query string) [] [3]string {
    result := [] [3]string {}
    query = strings.Replace(query, "-", " ", -1)
    terms := strset.MakeFromText(strings.ToUpper(query))
    scanner := bufio.NewScanner(text)
    for scanner.Scan() {
        line := scanner.Text()
        if strings.TrimSpace(line) == "" {
            continue
        }
        char, name, nameWords := ParseLine(line)
        if terms.SubsetOf(nameWords) {
            result = append(result,
                [3]string{fmt.Sprintf("U+%04X", char),
                          string(char), name})
        }
    }
    return result
}
```

ThoughtWorks®

CONCLUSÃO

CONJUNTOS NA PRÁTICA

Operações de conjunto podem simplificar sua lógica

`golang-set` (@deckarep): a implementação *rica* mais popular

`gen` pode produzir um `golang-set` específico para seu tipo

Codar um tipo set é um ótimo exercício de programação

Se precisar de `Set` para elementos string, pode usar `strset`

<https://github.com/standupdev/strset>

FICÇÃO CIENTÍFICA

Bom mesmo seria isso...

```
var s1 set[string]
```

```
s2 := make(set[uint64])
```

MUITO GRATO!

Luciano Ramalho

@ramalhoorg | @standupdev

luciano.ramalho@thoughtworks.com

ThoughtWorks®