

Goクイズで学ぶメソッドセット

Go言語仕様輪読会

2021/04/15

task4233

自己紹介

task4233 (Takashi MIMA)

趣味と実益を兼ねてGoを書いています

Go本体へのコントリビュート経験があります (<https://go-review.googlesource.com/c/go/+288472>)

The screenshot shows a web interface for reviewing a change in the Go project. At the top, there's a navigation bar with links for Go, CHANGES, YOUR, DOCUMENTATION, and BROWSE, along with a search bar and icons for help, settings, and a globe. Below the navigation bar, a green 'Merged' badge is visible, followed by the change ID 'fca94ab' and a star icon with the number '288472'. The title of the change is 'spec: improve the example in Type assertions section'. A 'REVERT' button is on the right. The main content area is divided into two columns. The left column contains metadata: 'Updated' (Feb 03), 'Owner' (Gerrit Bot), 'Uploader' (Robert Griesemer), 'Author' (task4233), 'Reviewers' (Rob Pike, Russ Cox, Ian Lance Taylor, Robert Griesemer), 'CC' (T M, Go Bot), 'Repo | Branch' (go | master), 'Parent' (98f8454), 'Topic' (No topic), and 'Hashtags' (no-owners). The right column contains a 'REPLY' button and a text box with the change description: 'spec: improve the example in Type assertions section'. Below the text box, there's a detailed description of the change: 'The example, var v, ok T1 = x.(T), can be interpreted as type T1 interface{} or type T = bool; type T1 = T. Separating the example would help understanding for readers.' This is followed by a list of links and information: 'Change-Id: I179f4564e67f4d503815d29307df2cebb50c82f9', 'GitHub-Last-Rev: b34fffb6bb07cb2883bc313ef3bc9980b3dd4abe', 'GitHub-Pull-Request: golang/go#44040', 'Reviewed-on: https://go-review.googlesource.com/c/go/+288472', 'Reviewed-by: Robert Griesemer <gri@golang.org>', 'Reviewed-by: Rob Pike <r@golang.org>', 'Reviewed-by: Ian Lance Taylor <iant@golang.org>', and 'Trust: Robert Griesemer <gri@golang.org>'. At the bottom, there's a summary of reviews: 'Code-Review' with +2 votes from Robert Griesemer, Ian Lance Taylor, and Rob Pike, and 'Trusted' with a thumbs up from Go Bot.

Go CHANGES YOUR DOCUMENTATION BROWSE

Merged as fca94ab ☆ 288472 spec: improve the example in Type assertions section REVERT

Updated Feb 03

Owner Gerrit Bot

Uploader Robert Griesemer

Author task4233

Reviewers Rob Pike Russ Cox X Ian Lance Tayl... Robert Griesemer

ADD REVIEWER

CC T M X Go Bot

ADD CC

Repo | Branch go | master

Parent 98f8454

Topic No topic

Hashtags no-owners

REPLY

spec: improve the example in Type assertions section

The example, var v, ok T1 = x.(T), can be interpreted as type T1 interface{} or type T = bool; type T1 = T. Separating the example would help understanding for readers.

Change-Id: I179f4564e67f4d503815d29307df2cebb50c82f9

GitHub-Last-Rev: b34fffb6bb07cb2883bc313ef3bc9980b3dd4abe

GitHub-Pull-Request: golang/go#44040

Reviewed-on: https://go-review.googlesource.com/c/go/+288472

Reviewed-by: Robert Griesemer <gri@golang.org>

Reviewed-by: Rob Pike <r@golang.org>

Reviewed-by: Ian Lance Taylor <iant@golang.org>

Trust: Robert Griesemer <gri@golang.org>

✓ Code-Review +2 Robert Griesemer +2 Ian Lance Tayl... +2 Rob Pike

✓ Trusted 👍 Go Bot

ゴールと理解するメリット

ゴール

interfaceを「実装する」という概念を通して、**メソッドセットとは何なのか理解する**

メリット

Goの仕様を理解する助けになる

(特にMethod callsやInterface typesにおける実装の部分)

おさらい：メソッドとは

- レシーバを持つ関数
- レシーバになれるのは、defined typeもしくはdefined typeのポインタ型

```
func (t *T) SayHello (x int) int {
```

レシーバ

メソッド名

シグネチャ

```
// 中身の処理
```

```
}
```

2種類のレシーバ：ポインタレシーバと値レシーバ

- ポインタレシーバとは、 **ポインタ型のレシーバ** のこと
- 値レシーバとは、 **値型のレシーバ** のこと

```
package main

import "fmt"

type List []int

// List型のレシーバなので値レシーバ
func (l List) AppendWithValueReceiver(num int) { l = append(l, num) }

// *List型のレシーバなのでポインタレシーバ
func (l *List) AppendWithPointerReceiver(num int) { *l = append(*l, num) }

func main() {
    l := List{1, 3, 5}
    l.AppendWithValueReceiver(7)
    fmt.Println(l)

    l.AppendWithPointerReceiver(7)
    fmt.Println(l)
}
```

Run

メソッドセットとは

メソッドセットは、型に関連付けられた **メソッドの集合** のこと

```
type Num int

// 型 Num のメソッドセットは
// - addOneWithValueReceiver
// - addWithValueReceiver
func (num Num) addOneWithValueReceiver() { num++ }

func (num Num) addWithValueReceiver(val int) { num += val }

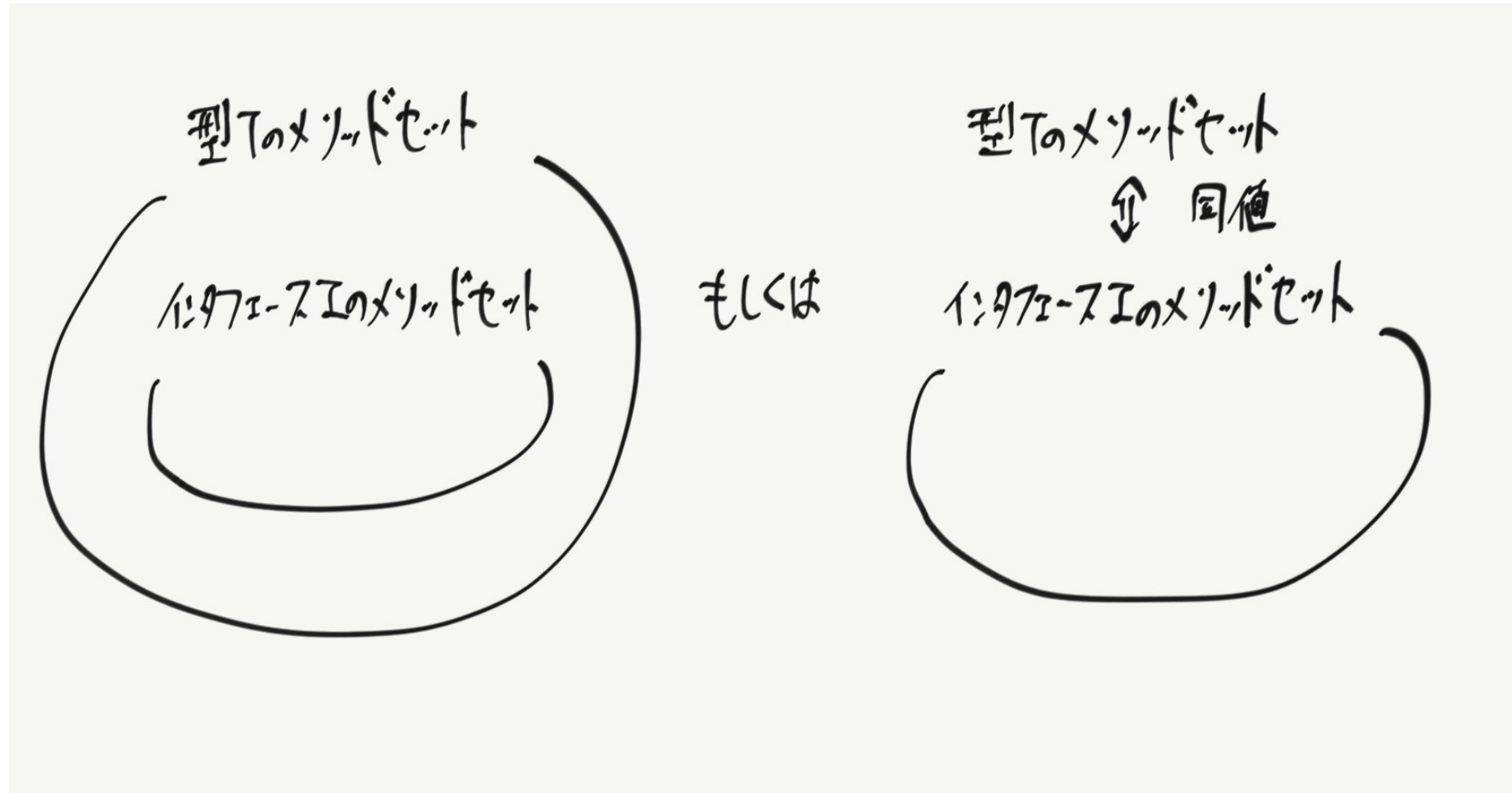
// 型 *Numのメソッドセットは
// - addOneWithPointerReceiver
// - addWithPointerReceiver
// - addOneWithValueReceiver
// - addWithValueReceiver
func (num *Num) addOneWithPointerReceiver() { *num++ }

func (num *Num) addWithPointerReceiver(val int) { *num += val }
```

Run

型Tがinterface Iを「実装する」とは

型 T がinterface I のメソッドセットを全て含むこと



メソッドセットの詳細な定義

- interface型のメソッドセットは、そのinterface定義
- 型 T のメソッドセットは、レシーバ型 T で宣言された全てのメソッドセット
- 型 $*T$ のメソッドセットは、レシーバ型 T または $*T$ で宣言された全てのメソッドセット
- 埋め込みフィールドを持つ構造体については、更なる規則が適用される(今回は時間が足りないので割愛)
- それ以外の型は、空のメソッドセットを持つ

interface型のメソッドセット

```
package main

type Animal interface {
    MakeSound() string
}

type Cat struct {
}

func (Cat) MakeSound() string {
    return "meow"
}

func main() {
    // Cat構造体はAnimalインタフェースを実装している
    var _ Animal = Cat{}
}
```

Run

Goクイズ - interface型のメソッドセット

```
package main

type Animal interface {
    MakeSound() string
}

type Cat struct {
}

func (Cat) MakeSound() []byte {
    return []byte("meow")
}

func main() {
    // Cat構造体はAnimalインタフェースを実装している?
    var _ Animal = Cat{}
}
```

Run

interface型のメソッドセット - 解答と解説

- 返り値が異なっているのでメソッドセットが異なるから実装していない
- 仮引数リストが異なる場合も同様

```
package main

type Animal interface {
    MakeSound() string
}

type Cat struct {
}

func (Cat) MakeSound() []byte {
    return []byte("meow")
}

func main() {
    // Cat構造体はAnimalインタフェースを実装していない
    // var _ Animal = Cat{}
}
```

Run

値レシーバに関するinterfaceの実装

```
package main

import "fmt"

type error interface {
    Error() string
}

type EmptyError struct {
    FieldName string
}

func (e *EmptyError) Error() string {
    return fmt.Sprintf("%s is empty", e.FieldName)
}

func main() {
    // EmptyError型は、Error メソッドを実装している？
    var _ error = EmptyError{}
}
```

Run

Goクイズ - 値レシーバに関するinterfaceの実装

```
package main

import "fmt"

type error interface {
    Error() string
}

type EmptyError struct {
    FieldName string
}

func (e *EmptyError) Error() string {
    return fmt.Sprintf("%s is empty", e.FieldName)
}

func main() {
    // EmptyError型は、Error メソッドを実装している？
    var _ error = EmptyError{}
}
```

Run

値レシーバに関するinterfaceの実装 - 解答と解説

```
package main

import "fmt"

type error interface {
    Error() string
}

type EmptyError struct {
    FieldName string
}

func (e *EmptyError) Error() string {
    return fmt.Sprintf("%s is empty", e.FieldName)
}

func main() {
    // EmptyError型は、Error メソッドを実装していない
    // var _ error = EmptyError{}

    // 実装したいなら型を合わせる必要がある
    var _ error = &EmptyError{}
}
```

Run

コラム: メソッド呼び出し時の特別ルール

x.m()というメソッド呼び出しは、xがaddressableで&xのメソッドセットがmを含んでいる場合、x.m()は(&x).m()の省略形になる

[Address operators - The Go Programming Language Specification](https://golang.org/ref/spec#Address_operators) (https://golang.org/ref/spec#Address_operators)

```
package main

import "fmt"

type EmptyError struct {
    fieldName string
}

func (e EmptyError) Error() string {
    return fmt.Sprintf("%s is empty", e.FieldName)
}

func main() {
    var emptyError EmptyError = EmptyError{FieldName: "hoge"}

    // (&emptyError).Error() と同義
    fmt.Println(emptyError.Error())
}
```

Run

ポインタレシーバに関するinterfaceの実装

```
package main

import "fmt"

type error interface {
    Error() string
}

type EmptyError struct {
    FieldName string
}

func (e *EmptyError) Error() string {
    return fmt.Sprintf("%s is empty", e.FieldName)
}

func main() {
    // EmptyError型は、Error メソッドを実装している？
    var _ error = &EmptyError{}
}
```

Run

Goクイズ - ポインタレシーバに関するinterfaceの実装

```
package main

import "fmt"

type error interface {
    Error() string
}

type EmptyError struct {
    FieldName string
}

func (e EmptyError) Error() string {
    return fmt.Sprintf("%s is empty", e.FieldName)
}

func main() {
    // EmptyError型は、Error メソッドを実装している？
    var _ error = &EmptyError{}
}
```

Run

ポインタレシーバに関するinterfaceの実装 - 解答と解説

```
package main

import "fmt"

type error interface {
    Error() string
}

type EmptyError struct {
    FieldName string
}

func (e *EmptyError) Error() string {
    return fmt.Sprintf("%s is empty", e.FieldName)
}

func main() {
    // EmptyError型は、Error メソッドを実装していない
    // var _ error = EmptyError{}

    // 実装したいなら型を合わせる必要がある
    var _ error = &EmptyError{}
}
```

Run

コラム: interfaceの実装

```
package main_test

import "context"

// UserRepository がUserUseCaseを実装していることをコンパイル時に保証する
var _ UserRepository = (*UserUseCase)(nil)

type UserRepository interface {
    Create(ctx context.Context, name string) error
}

type UserUseCase struct {
    userRepo UserRepository
}

func (u *UserUseCase) Create(ctx context.Context, name string) error {
    // TODO: ユーザを作成する
    return nil
}

func main() {
}
```

Run

空のinterface

```
package main

import "fmt"

type EmptyInterface interface {
}

type T struct {
}

func (T) Hello() {
    fmt.Println("Hello!")
}

func main() {
    // EmptyInterface のメソッドセットはなく、Tのメソッド
    var _ EmptyInterface = T{}
}
```

Run

Goクイズ - 空のinterface

```
package main

type EmptyInterface interface{}

func main() {
    var _ EmptyInterface = nil
}
```

Run

空のinterface - 解答と解説

- interface 以外の型は、デフォルトで空のメソッドセットを持つ
- 空のinterfaceは何もメソッドを含まないので、全ての値を代入可能

```
package main

type EmptyInterface interface{}

func main() {
    // 全ての値を代入可能
    var _ EmptyInterface = nil
    var _ EmptyInterface = 57
    var _ EmptyInterface = "hoge"

    type Person struct {
        Name string
    }
    var _ EmptyInterface = Person{}
}
```

Run

コラム: 空のinterfaceへの代入

```
package main

func main() {
    var _ interface{} = nil

    var Num interface{} = -1

    // 型が異なるので実装できない
    // var _ int = Num + 1

    // Underlying typeが異なるのでConversionできない
    // var _ int = int(Num) + 1

    // type assertionすればOK
    var _ int = Num.(int) + 1
}
```

Run

まとめ

- interface型のメソッドセットは、そのinterface定義
- 型 T のメソッドセットは、レシーバ型 T で宣言された全てのメソッドセット
- 型 $*T$ のメソッドセットは、レシーバ型 T または $*T$ で宣言された全てのメソッドセット
- 埋め込みフィールドを持つ構造体については、更なる規則が適用される(今回は時間が足りないので割愛)
- それ以外の型は、空のメソッドセットを持つ

Thank you

Go言語仕様輪読会

2021/04/15

task4233

[@task4233](http://twitter.com/task4233) (<http://twitter.com/task4233>)

