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

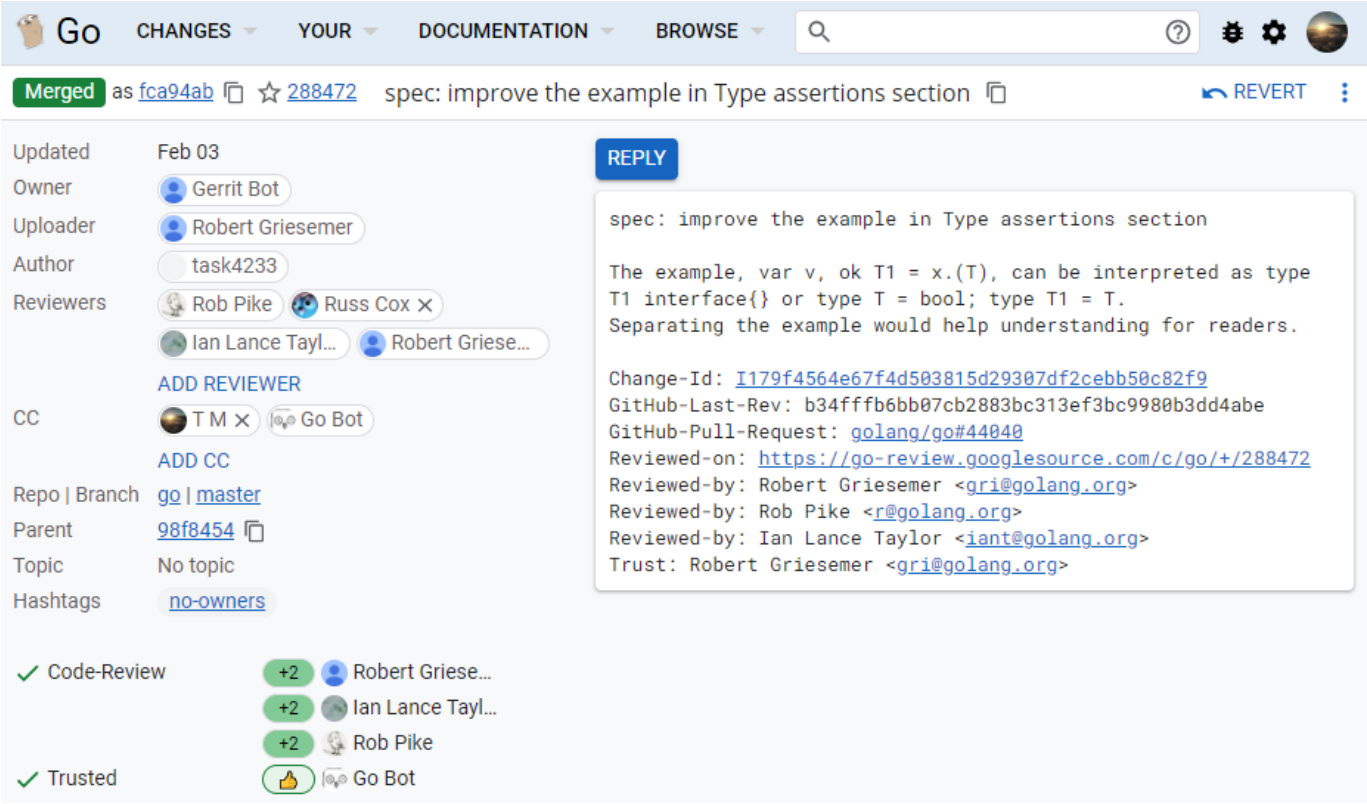
task4233

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)



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 {
    // 中身の処理
}
```

レシーバ　メソッド名　シグネチャ

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 Num 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() {
    num := Num(2)
    num.addOneWithValue()
    fmt.Println(num)

    num.addOneWithPointer()
    fmt.Println(num)
}
```

Run

メソッドセットとは

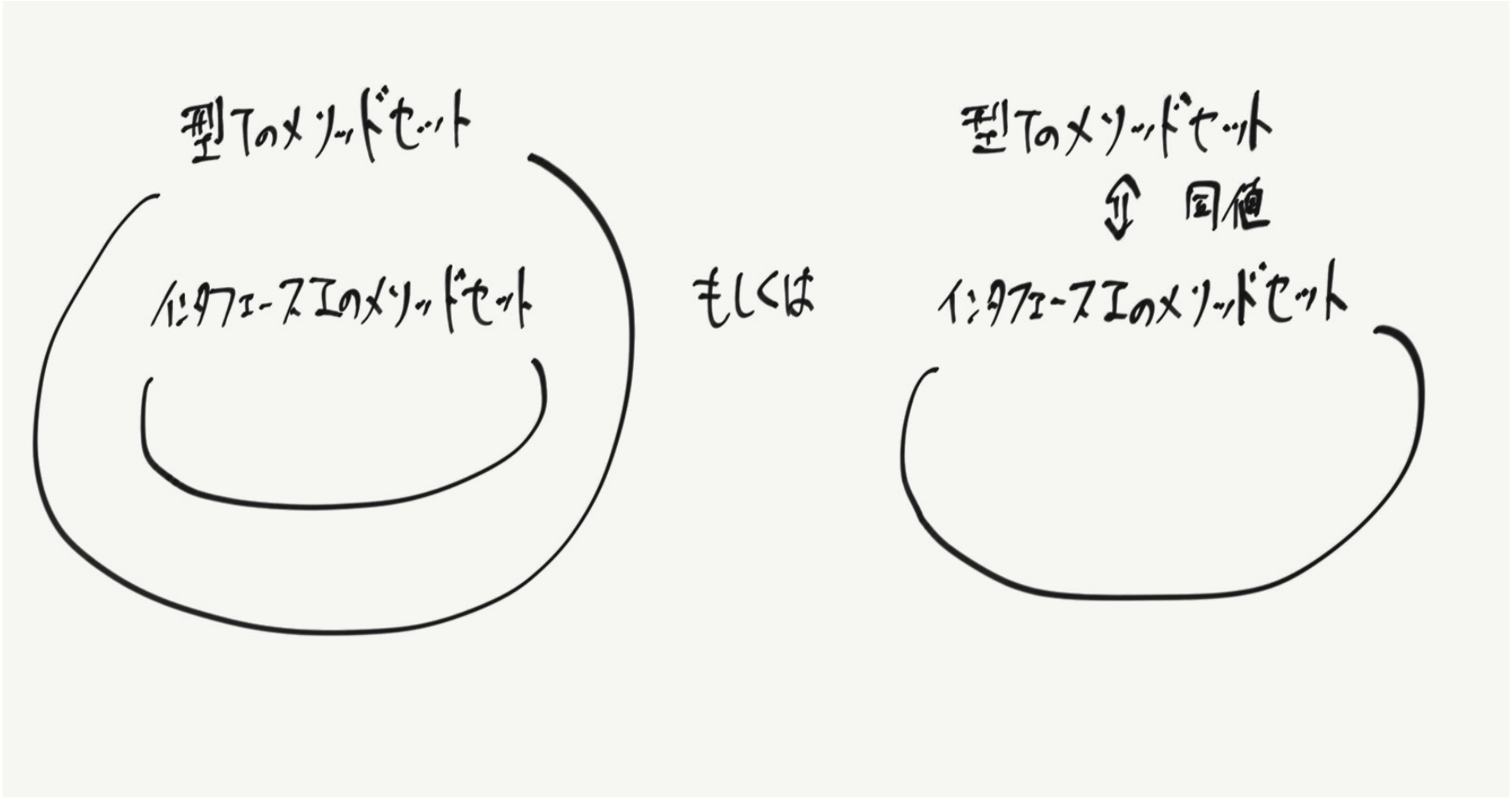
メソッドセットは、型に関連付けられた**メソッドの集合**のこと

```
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interfaceIを「実装する」とは

型TがinterfaceI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定義で列挙されるメソッドの集合
- 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の集合
- 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の集合
- 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型はAnimal interface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型はAnimal interface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型はAnimal interface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)

```
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

import "context"

// *UserUseCase型がUserRepository interface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

    // Numと1の型が異なるので、invalidな式の演算が原因でcompile errorになる
    // var _ int = Num + 1

    // intとNumのUnderlying typeが異なるので、Coversionできないことが原因でcompile errorになる
    // 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定義で列挙されるメソッドの集合
- 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の集合
- 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の集合
- 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)