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

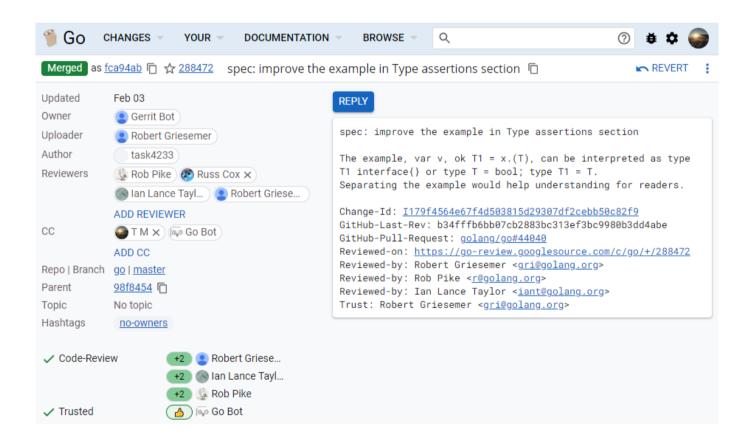
task4233

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

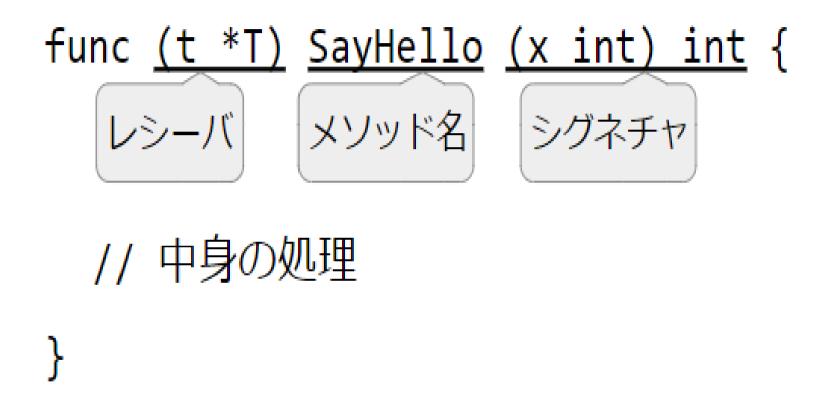
interfaceを「実装する」という概念を通して、メソッドセットとは何なのか理解する

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

defined typeかdefined typeを指すポインタを、メソッドのレシーバとして持つ関数



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(参照渡し)
- 値レシーバとは、値型のレシーバのこと(値渡し)

```
package main
import "fmt"
type Num int
func (num Num) addOneWithValue() { num++ }
func (num *Num) addOneWithPointer() { *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 List []int

// 型 List のメソッドセットはAppendWithValueReceiver
func (1 List) AppendWithValueReceiver(num int) { 1 = append(1, num) }

// 型 *List のメソッドセットはAppendWithPointerReceiver
func (1 *List) AppendWithPointerReceiver(num int) { *1 = append(*1, num) }

func main() {
}
```

型Tがinterface Iを「実装する」とは

型 T のメソッドセットがinterface I のメソッドセットを全て含むこと

(ここにmethod sets of I c method sets of `T`の包含関係を示す図が入る)

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
- 埋め込み型については、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構造体は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

Addressable (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UserUsecase 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
}
```

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{}
}
```

コラム: 空のinterfaceへの代入

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

    var Num interface{} = -1

    // Numと1の型が異なるのでinvalidな式の評価になる
    // var _ int = Num + 1

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

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

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
- 埋め込み型については、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)