Golang 中的接口

主讲教师：（大地）
合作网站：www.itying.com （IT 营）
我的专栏：https://www.itying.com/category-79-b0.html

一、接口的介绍

1、现实生活中的接口

现实生活中手机、相机、U 盘都可以和电脑的 USB 接口建立连接。我们不需要关注 usb 卡槽大小是否一样，因为所有的 USB 接口都是按照统一的标准来设计的。


2、Golang 中的接口（interface）

Golang 中的接口是一种抽象数据类型，Golang 中接口定义了对象的行为规范，只定义规范不实现。接口中定义的规范由具体的对象来实现。

通俗的讲接口就一个标准，它是对一个对象的行为和规范进行约定，约定实现接口的对象必须得按照接口的规范。

二、Golang 接口的定义

在 Golang 中接口（interface）是一种类型，一种抽象的类型。接口（interface）是一组函数 method 的集合，Golang 中的接口不能包含任何变量。

在 Golang 中接口中的所有方法都没有方法体，接口定义了一个对象的行为规范，只定义规范不实现。接口体现了程序设计的多态和高内聚低耦合的思想

Golang 中的接口也是一种数据类型，不需要显示实现。只需要一个变量含有接口类型中的所有方法，那么这个变量就实现了这个接口。

Golang 中每个接口由数个方法组成，接口的定义格式如下：

```
type 接口名 interface{
    方法名 1( 参数列表 1 ) 返回值列表 1
    方法名 2( 参数列表 2 ) 返回值列表 2
    …
}
```

其中：

- **接口名**：使用 type 将接口定义为自定义的类型名。Go 语言的接口在命名时，一般会在单词后面添加 er，如有写操作的接口叫 Writer，有字符串功能的接口叫 Stringer 等。接口名最好要能突出该接口的类型含义。
- **方法名**：当方法名首字母是大写且这个接口类型名首字母也是大写时，这个方法可以被接口所在的包（package）之外的代码访问。
- **参数列表、返回值列表**：参数列表和返回值列表中的参数变量名可以省略。

演示：定义一个 Usber 接口让 Phone 和 Camera 结构体实现这个接口

```
package main


import "fmt"


type Usber interface {
```

```
        Start()
        Stop()
}

type Phone struct {
        Name string
}

func (p Phone) Start() {
        fmt.Println(p.Name, "开始工作")
}
func (p Phone) Stop() {
        fmt.Println("phone  停止")
}

type Camera struct {
}

func (c Camera) Start() {
        fmt.Println("相机  开始工作")
}
func (c Camera) Stop() {
        fmt.Println("相机  停止工作")
}

func main() {
```

```
    phone := Phone{
        Name: "小米手机",
    }
    var p Usber = phone //phone 实现了 Usb 接口
    p.Start()

    camera := Camera{}
    var c Usber = camera //camera 实现了 Usb 接口
    c.Start()
}
```

演示：Computer 结构体中的 Work 方法必须传入一个 Usb 的接口

```
package main

import "fmt"

type Usber interface {
    Start()
    Stop()
}

type Phone struct {
    Name string
}
```

```
func (p Phone) Start() {
	fmt.Println(p.Name, "开始工作")
}
func (p Phone) Stop() {
	fmt.Println("phone  停止")
}

type Camera struct {
}

func (c Camera) Start() {
	fmt.Println("相机  开始工作")
}
func (c Camera) Stop() {
	fmt.Println("相机  停止工作")
}

//电脑的结构体
type Computer struct {
	Name string
}
//  电脑的 Work 方法要求必须传入 Usb 接口类型数据
func (c Computer) Work(usb Usber) {
	usb.Start()
	usb.Stop()
}
func main() {
```

```
    phone := Phone{
        Name: "小米手机",
    }
    camera := Camera{}
    computer := Computer{}
    //把手机插入电脑的 Usb 接口开始工作
    computer.Work(phone)
    //把相机插入电脑的 Usb 接口开始工作
    computer.Work(camera)
}
```

三、空接口

Golang 中的接口可以不定义任何方法，没有定义任何方法的接口就是空接口。空接口表示没有任何约束，因此任何类型变量都可以实现空接口。

空接口在实际项目中用的是非常多的，用空接口可以表示任意数据类型。

案例：

```
func main() {
    // 定义一个空接口 x, x 变量可以接收任意的数据类型
    var x interface{}
    s := "你好 golang"
    x = s
    fmt.Printf("type:%T value:%v\n", x, x)

    i := 100
    x = i
    fmt.Printf("type:%T value:%v\n", x, x)

    b := true
    x = b
    fmt.Printf("type:%T value:%v\n", x, x)

}
```

1、空接口作为函数的参数

使用空接口实现可以接收任意类型的函数参数。

```
// 空接口作为函数参数
func show(a interface{}) {
    fmt.Printf("type:%T value:%v\n", a, a)
}
```

2、map 的值实现空接口

使用空接口实现可以保存任意值的字典。

```
// 空接口作为 map 值
var studentInfo = make(map[string]interface{})
studentInfo["name"] = "张三"
studentInfo["age"] = 18
studentInfo["married"] = false
fmt.Println(studentInfo)
```

3、切片实现空接口

```
var slice = []interface{}{"张三", 20, true, 32.2}
fmt.Println(slice)
```

四、类型断言

一个接口的值（简称接口值）是由一个具体类型和具体类型的值两部分组成的。这两部分分别称为接口的动态类型和动态值。

如果我们想要判断空接口中值的类型，那么这个时候就可以使用类型断言，其语法格式：

```
x.(T)
```

其中：

- x：表示类型为 interface{}的变量
- T：表示断言 x 可能是的类型。

该语法返回两个参数，第一个参数是 x 转化为 T 类型后的变量，第二个值是一个布尔值，若

为 true 则表示断言成功，为 false 则表示断言失败。

举个例子：

```
func main() {
    var x interface{}
    x = "Hello golnag"
    v, ok := x.(string)
    if ok {
        fmt.Println(v)
    } else {
        fmt.Println("类型断言失败")
    }
}
```

上面的示例中如果要断言多次就需要写多个 if 判断，这个时候我们可以使用 switch 语句来实现：

注意：类型.(type)只能结合 switch 语句使用

```
func justifyType(x interface{}) {
    switch v := x.(type) {
    case string:
        fmt.Printf("x is a string，value is %v\n", v)
    case int:
        fmt.Printf("x is a int is %v\n", v)
    case bool:
        fmt.Printf("x is a bool is %v\n", v)
    default:
        fmt.Println("unsupport type！")
    }
}
```

因为空接口可以存储任意类型值的特点，所以空接口在 Go 语言中的使用十分广泛。

关于接口需要注意的是：只有当有两个或两个以上的具体类型必须以相同的方式进行处理时才需要定义接口。不要为了接口而写接口，那样只会增加不必要的抽象，导致不必要的运行时损耗。

五、结构体值接收者和指针接收者实现接口的区别

值接收者：

如果结构体中的方法是值接收者，那么实例化后的结构体值类型和结构体指针类型都可以赋值给接口变量

```
package main
import "fmt"
type Usb interface {
	Start()
	Stop()
}
type Phone struct {
	Name string
}
func (p Phone) Start() {
	fmt.Println(p.Name, "开始工作")
}
func (p Phone) Stop() {
	fmt.Println("phone  停止")
}
func main() {

	phone1 := Phone{
		Name: "小米手机",
	}
	var p1 Usb = phone1 //phone1 实现了 Usb 接口    phone1 是  Phone 类型
	p1.Start()                //小米手机  开始工作

	phone2 := &Phone{
		Name: "苹果手机",
	}
	var p2 Usb = phone2 //phone2 实现了 Usb 接口    phone2 是  *Phone 类型
	p2.Start()                //苹果手机  开始工作

}
```

指针接收者：

如果结构体中的方法是指针接收者，那么实例化后结构体指针类型都可以赋值给接口变量，结构体值类型没法赋值给接口变量。

```
package main
import "fmt"
type Usb interface {
	Start()
	Stop()
}
type Phone struct {
	Name string
}
func (p *Phone) Start() {
	fmt.Println(p.Name, "开始工作")
}
func (p *Phone) Stop() {
	fmt.Println("phone  停止")
}

func main() {
	/*
		错误写法
			phone1 := Phone{
				Name: "小米手机",
			}
			var p1 Usb = phone1
			p1.Start()
	*/
	//正确写法
	phone2 := &Phone{
		Name: "苹果手机",
	}
	var p2 Usb = phone2 //phone2 实现了 Usb 接口    phone2 是  *Phone 类型
	p2.Start()                //苹果手机  开始工作
}
```

六、一个结构体实现多个接口

Golang 中一个结构体也可以实现多个接口

```
package main

import "fmt"
```

```
type AInterface interface {
	GetInfo() string
}

type BInterface interface {
	SetInfo(string, int)
}

type People struct {
	Name string
	Age  int
}

func (p People) GetInfo() string {
	return fmt.Sprintf("姓名:%v  年龄:%d", p.Name, p.Age)
}

func (p *People) SetInfo(name string, age int) {
	p.Name = name
	p.Age = age
}

func main() {

	var people = &People{
		Name: "张三",
		Age:  20,
	}

	// people 实现了 AInterface 和 BInterface
	var p1 AInterface = people
	var p2 BInterface = people

	fmt.Println(p1.GetInfo())
	p2.SetInfo("李四", 30)
	fmt.Println(p1.GetInfo())
}
```

People的两个方法实现

初始化People结构体

将people转化为Ainterface接口，并赋值给p1
将people转化为Binterface接口，并赋值给p2

七、接口嵌套

接口与接口间可以通过嵌套创造出新的接口。

```
package main
import "fmt"
type SayInterface interface {
	say()
}
type MoveInterface interface {
	move()
}
// 接口嵌套
type Animal interface {
	SayInterface
	MoveInterface
}
type Cat struct {
	name string
}

func (c Cat) say() {
	fmt.Println("喵喵喵")
}

func (c Cat) move() {
	fmt.Println("猫会动")
}

func main() {
	var x Animal
	x = Cat{name: "花花"}
	x.move()
	x.say()
}
```