Go 语言从入门到项目实战

第10章 反射

10.1 使用反射访问变量

- 使用 reflect 包
  - 获取变量类型：func TypeOf(i interface{}) Type
  - 获取变量的值：func ValueOf(i interface{}) Value
  - 反射值的非空判定：func (v Value) IsNil() bool
  - 反射值的有效性判定：func (v Value) IsValid() bool

10.1 使用反射访问变量

- 示例

```
func main() {
        var intNum int
        fmt.Println(reflect.TypeOf(intNum))
}
```

10.1 使用反射访问变量

- 示例

```
func main() {
        var intNum int
        typeOfIntNum := reflect.TypeOf(intNum)
        fmt.Println(typeOfIntNum.Name(), typeOfIntNum.Kind())
        type structType struct {}
        typeOfStructType:=reflect.TypeOf(structType{})
        fmt.Println(typeOfStructType.Name(), typeOfStructType.Kind())
}
```

10.1 使用反射访问变量

- 示例

```
func main() {
        var intNum int = 100
        fmt.Println(reflect.ValueOf(intNum))
}
```

10.1 使用反射访问变量

- 示例

```
func main() {
    var intNum int64 = 100
    valueOfIntNum := reflect.ValueOf(intNum)
    //类型强制转换为Int
    var originIntNum int64 = int64(valueOfIntNum.Int())
    fmt.Println(originIntNum)
}
```

10.1 使用反射访问变量

- 示例

```
func main() {
    var intNum int64 = 100
    valueOfIntNum := reflect.ValueOf(intNum)
    //类型强制转换为Int
    var originIntNum int64 = valueOfIntNum.Interface().(int64)
    fmt.Println(originIntNum)
}
```

10.1 使用反射访问变量

- 示例

```
func main() {
    var intNums = []int{1,3,5,7,9}
    valueOfIntNum := reflect.ValueOf(intNums)
    fmt.Println(valueOfIntNum.IsNil())
    fmt.Println(valueOfIntNum.IsValid())
}
```

10.2 使用反射访问指针表示的变量

```
func main() {
    intNum := 200
    ins := &intNum
    typeOfIntNumPtr := reflect.TypeOf(ins)                    //获取指针变量的类型名和种类
    fmt.Println(typeOfIntNumPtr.Name(), typeOfIntNumPtr.Kind())
    typeOfIntNumPtr = typeOfIntNumPtr.Elem()          //获取指针变量所表示的变量的类型名和种类
    fmt.Println(typeOfIntNumPtr.Name(), typeOfIntNumPtr.Kind())
}
```

10.3 使用反射访问结构体

```
func main() {
    type cat struct {
        name string `meaning:"全名"`
        age  int   `meaning:"年龄"`
    }
    catOne := cat{name: "三酷猫", age: 18}   //声明catOne类型变量并赋初始值
    typeOfCat := reflect.TypeOf(catOne)            //通过反射获取catOne变量的类型名和种类
    fmt.Println(typeOfCat.Name(), typeOfCat.Kind())
```

10.3 使用反射访问结构体

```
    valueOfCat := reflect.ValueOf(catOne)                    //通过反射获取catOne变量的值
        for i := 0; i < typeOfCat.NumField(); i++ {
        fieldType := typeOfCat.Field(i)                      //通过反射获取结构体成员的名称和值
        fmt.Println(fieldType.Index, fieldType.Name, valueOfCat.Field(i), fieldType.Tag)
    }
    catType, ok := typeOfCat.FieldByName("age")                  //查找名为age的成员
    if ok {
        fmt.Println(catType.Tag.Get("meaning"))      //输出age成员的部分Tag文本

    }
}
```

10.4 通过反射修改值

函 数 名	作 用	函数声明
Elem()	相当于*操作。 取某个地址变量表示的值。当该函数作用于非指针或接口时将发生宕机，作用于空指针时返回nil。	func (v Value) Elem() Value
Addr()	相当于&操作。 取某个值的地址。当该函数作用于不可寻址的值时将发生宕机。	func (v Value) Addr() Value
CanAddr()	判断某个值是否可寻址，可寻址将返回true，反之则返回false。	func (v Value) CanAddr() bool
CanSet()	判断某个值是否可修改，可修改将返回true，反之则返回false。	func (v Value) CanSet() bool

10.4 通过反射修改值

- 示例

```
func main() {
        numA := 100
        addrValueOfNumA := reflect.ValueOf(&numA)        //取numA的地址
        fmt.Println(&numA)
        valueOfNumA := addrValueOfNumA.Elem()            //获取地址所表示的值，即numA的值
        fmt.Println(valueOfNumA.Addr())                  //获取valueOfNumA所在地址
        fmt.Println(valueOfNumA.CanAddr())               //判断该值是否可被寻址
}
```

10.4 通过反射修改值

函 数 名	作　用	函数声明
SetInt()	修改int类型值，允许作用于类型为int、 int8、int16、 int32和int64的值。	func (v Value) SetInt(x int64)
SetUInt()	修改uint类型值，允许作用于类型为uint、uint8、uint16、uint32和uint64的值。	func (v Value) SetUint(x uint64)
SetFloat()	修改float类型值，允许作用于类型为float32和float64的值。	func (v Value) SetFloat(x float64)
SetBool()	修改bool类型值，允许作用于类型为bool的值。	func (v Value) SetBool(x bool)
SetBytes()	修改[]bytes类型值，允许作用于类型为[]bytes的值。	func (v Value) SetBytes(x []byte)
SetString()	修改string类型值，允许作用于类型为string的值。	func (v Value) SetString(x string)

10.4 通过反射修改值

- 示例

```
func main() {
        numA := 100
        addrValueOfNumA := reflect.ValueOf(&numA) //取numA的地址
        valueOfNumA := addrValueOfNumA.Elem()     //获取地址所表示的值，即numA的值
        fmt.Println(valueOfNumA.CanAddr())        //判断该值是否可被寻址
        valueOfNumA.SetInt(200)                   //修改numA的值为200
        fmt.Println(numA)                         //输出numA变量的值
}
```

10.4 通过反射修改值

- 示例

```
func main() {
        type cat struct {Name string
                age  int}
        catA := &cat{Name: "姓名", age: 18}                  //声明变量catA，类型为cat，并赋初始值
        fmt.Println(catA)                                    //输出catA的值
        addrValueOfCat := reflect.ValueOf(catA)              //取catA的地址
        valueOfCat := addrValueOfCat.Elem()                  //通过addrValueOfCat变量获取catA的值
        catName := valueOfCat.FieldByName("Name")            //找到名为Name的成员变量
        catName.SetString("三酷猫")                           //修改名为Name的成员变量值
        fmt.Println(catA)                                    //输出catA的值
}
```

10.5 通过反射调用函数

- 示例

```
func main() {
        sayHelloValue := reflect.ValueOf(sayHello)                                          //将函数包装为反射值变量
        sayHelloParam := []reflect.Value{reflect.ValueOf("三酷猫"), reflect.ValueOf("你好呀！")}    //定义函数传入的参数切片
        results := sayHelloValue.Call(sayHelloParam)                                        //通过反射调用函数
        fmt.Println(results)                                                                //输出函数执行结果
}
func sayHello(name string, content string) string {
        output := name + "说：" + content
        return output
}
```

10.6 使用反射创建变量

- 示例

```
func main() {
    var numA int                                        //声明int型变量numA
    typeOfNumA := reflect.TypeOf(numA)                  //获取numA的类型，并存入变量typeOfNumA
    numAVal := reflect.New(typeOfNumA)                  //根据类型创建变量numAVal
    fmt.Println(numAVal, numAVal.Type(), numAVal.Kind()) //输出numAVal的值、类型名称和种类名称
    numAVal.Elem().SetInt(100)                          //修改numAVal表示的值
    fmt.Println(numAVal.Elem().Int())                   //输出numAVal表示的值
}
```