结构体

自定义类型和类型别名

```go
type myint int//自定义类型
type yourint = int//类型别名

func main() {
    var n myint = 89
    var m yourint = 10
    fmt.Printf("%T\n", n)
    fmt.Printf("%T\n", m)
}
```

程序运行结果：

```
<4 go 设置调用>
main.myint
int

进程 已完成，退出代码为 0
```

自定义类型在编译的时候，已经默认有这个类型了，但是类型别名在编译的时候还是原来的类型。

结构体

Go语言中的基础数据类型可以表示一些事物的基本属性，但是当我们想表达一个事物的全部或部分属性时，这时候再用单一的基本数据类型明显就无法满足需求了，Go语言提供了一种自定义数据类型，可以封装多个基本数据类型，这种数据类型叫结构体，英文名称 `struct` 。也就是我们可以通过 `struct` 来定义自己的类型了。

Go语言中通过 `struct` 来实现面向对象。

结构体的定义

使用 `type` 和 `struct` 关键字来定义结构体，具体代码格式如下：

```
type 类型名 struct {
    字段名 字段类型
    字段名 字段类型
    …
}
```

其中：

- 类型名：标识自定义结构体的名称，在同一个包内不能重复。
- 字段名：表示结构体字段名。结构体中的字段名必须唯一。
- 字段类型：表示结构体字段的具体类型。

示例 Go

```go
package main

import "fmt"
//定义的结构体
type person struct {
    name   string
    age    int
    gender string
    hoby   []string
}

func main() {
    var zhoulin person
    zhoulin.name = "Zhoulin"
    zhoulin.age = 20
    zhoulin.gender = "女"
    zhoulin.hoby = []string{"篮球", "足球", "双色球"}
    fmt.Println(zhoulin.name, zhoulin.age, zhoulin.gender, zhoulin.hoby)
}
```

在同一个结构体中的字段名不能重复。

匿名结构体：

匿名结构体 Go

```go
//这里的var直接将s声明出来，相当于已经实例化了
var s struct {
    x string
    y int
}
s.x = "hello"
s.y = 100
fmt.Println(s.x, s.y)
```

匿名结构体多用于临时场景。

结构体当成值传入函数 Go

```go
package main

import "fmt"

// 结构体是值类型
type person struct {
    name   string
    age    int
    gender string
    hoby   []string
}

func f(person2 person) {
    fmt.Println(person2.name)
    fmt.Println(person2.age)
    fmt.Println(person2.gender)
    fmt.Println(person2.hoby)
}

func main() {
    var zhoulin person
    zhoulin.name = "Zhoulin"
    zhoulin.age = 20
    zhoulin.gender = "女"
    zhoulin.hoby = []string{"篮球", "足球", "双色球"}
    f(zhoulin)
}
```

作为值传入时操作的只是一个“副本”，如果需要改变需要传地址。

创建指针类型结构体

```
var p2 = new(person)
fmt.Printf("%T\n", p2)
```

运行结果：

```
*main.person
```

发现p2是一个person类型的指针

结构体对象的初始化方式：

两种赋值 Go

```
p1 := person{
    name:   "john",
    age:    22,
    gender: "m",
    hoby:   []string{"james", "burgess", "milk"},
}
fmt.Println(p1)
p2 := person{
    "xiaowang",
    19,
    "男",
    []string{"james", "burgess", "milk"},
}
fmt.Println(p2)
```

模拟构造函数初始化

```go
func newperson(name string, age int) *person {
    return &person{
        name: name,
        age:  age,
    }
}

func main() {
    p := newperson("Alice", 30)
    fmt.Println(*p)
    p2 := newperson("Bob", 40)
    fmt.Println(*p2)
}
```

当结构体类型较少时可以直接返回结构体类型，当结构体类型较多时可以返回相应的指针类型。

结构体模拟继承

```go
package main

import "fmt"

type animal struct {
    name string
}

func (a animal) move() {
    fmt.Printf("%s moved animals\n", a.name)
}

type dog struct {
    feet uint8
    animal//animal拥有的方法dog此时也有了。
    //结构体的嵌套
}

func (d dog) wang() {
    fmt.Printf("%s汪汪汪", d.name)
}

func main() {
    d1 := dog{
        animal: animal{name: "周琳"},
        feet:   4,
    }
    fmt.Println(d1)
    d1.wang()
    d1.move()
}
```

结构体与JSON

序列化 Go

```go
package main

import (
    "encoding/json"
    "fmt"
)

//把go语言中的结构体变量--》json格式的字符串
//json的字符串--》Go语言中的结构体

type person struct {
    Name string
    Age  string
}

func main() {
    p1 := person{
        Name: "周琳",
        Age:  "18",
    }
    b, err := json.Marshal(p1)
    if err != nil {
        fmt.Printf("json marshal fail, err:%v\n", err)
        return
    }
    fmt.Printf("%#v\n", string(b))
}
```

json是前端使用的一种转义格式

注意：如果结构体内部变量名称是小写，则在json包中无法获取数据

所以要想输出有内容的结构，就需要在结构体中的变量名称用大写开头。

反序列化 Go

```go
var p2 person
str := `{"name":"理想","age":18}`
json.Unmarshal([]byte(str), &p2)
fmt.Printf("%#v\n", p2)
```

两端代码的运行结果：

```
<4 go 设置调用>
"{\"Name\":\"周琳\",\"Age\":\"18\"}"
main.person{Name:"理想", Age:""}

进程 已完成，退出代码为 0
```