

Golang 结构体和 Json 相互转换 序列化 反序列化

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、 关于 JSON 数据.....	1
二、 结构体与 JSON 序列化.....	2
三、 结构体标签 Tag.....	4
四、 嵌套结构体和 JSON 序列化反序列化.....	6
五、 关于 Map、切片的序列化反序列化.....	9

一、关于 JSON 数据

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式。易于人阅读和编写。同时也易于机器解析和生成。RESTfull Api 接口中返回的数据都是 json 数据。

Json 的基本格式如下：

```
{  
  
    "a": "Hello",  
  
    "b": "World"  
  
}
```

稍微复杂点的 JSON

```
{  
  
    "result": [{  
  
        "_id": "59f6ef443ce1fb0fb02c7a43",  
  
        "title": "笔记本电脑",  
  
    }  
  
}
```

```

    "status": "1",

    "pic": "public\\upload\\UObZahqPYzFvx_C9CQjU8KiX.png",

    "url": "12"

}, {

    "_id": "5a012efb93ec4d199c18d1b4",

    "title": "第二个轮播图",

    "status": "1",

    "pic": "public\\upload\\f3OtH11ZaPX5AA4Ov95Q7DEM.png"

}, {

    "_id": "5a012f2433574208841e0820",

    "title": "第三个轮播图",

    "status": "1",

    "pic": "public\\upload\\s5ujmYBQVRcLuvBHvWFMJHzS.jpg"

}, {

    "_id": "5a688a0ca6dcba0ff4861a3d",

    "title": "教程",

    "status": "1",

    "pic": "public\\upload\\Zh8EP9HOasV28ynDsp8TaGwd.png"

}}

}

```

二、结构体与 JSON 序列化

比如我们 Golang 要给 App 或者小程序提供 Api 接口数据，这个时候就需要涉及到结构体和 Json 之间的相互转换

Golang JSON 序列化是指把结构体数据转化成 JSON 格式的字符串，**Golang JSON 的反序列化**是指把 JSON 数据转化成 Golang 中的结构体对象

Golang 中的序列化和反序列化主要通过 "encoding/json" 包中的 json.Marshal() 和

json.Unmarshal()方法实现

1、结构体对象转化成 Json 字符串

```
package main

import (
    "encoding/json"
    "fmt"
)

type Student struct {
    ID      int
    Gender  string
    name    string //私有属性不能被 json 包访问
    Sno     string
}

func main() {

    var s1 = Student{
        ID:      1,
        Gender: "男",
        Name:    "李四",
        Sno:     "s0001",
    }

    fmt.Printf("%#v\n", s1)
    var s, _ = json.Marshal(s1)

    jsonStr := string(s)
    fmt.Println(jsonStr)
}
```

2、Json 字符串转换成结构体对象

```
package main
```

```
import (
    "encoding/json"
    "fmt"
)

type Student struct {
    ID      int
    Gender  string
    Name    string
    Sno     string
}

func main() {
    // var jsonStr = "{\"ID\":1,\"Gender\":\"男\",\"Name\":\"李四\",\"Sno\":\"s0001\"}"
    var jsonStr = `{"ID":1,"Gender":"男","Name":"李四","Sno":"s0001"}`
    //定义一个 Monster 实例
    var student Student
    err := json.Unmarshal([]byte(jsonStr), &student)
    if err != nil {
        fmt.Printf("unmarshal err=%v\n", err)
    }
    fmt.Printf("反序列化后 student=%#v student.Name=%v\n", student, student.Name)
}
```

三、结构体标签 Tag

Tag 是结构体的元信息，可以在运行的时候通过反射的机制读取出来。Tag 在结构体字段的后方定义，由一对反引号包裹起来，具体的格式如下：

```
`key1:"value1" key2:"value2"``
```

结构体 tag 由一个或多个键值对组成。键与值使用冒号分隔，值用双引号括起来。同一个结构体字段可以设置多个键值对 tag，不同的键值对之间使用空格分隔。

注意事项： 为结构体编写 Tag 时，必须严格遵守键值对的规则。结构体标签的解析代码的容错能力很差，一旦格式写错，编译和运行时都不会提示任何错误，通过反射也无法正确取值。例如不要在 key 和 value 之间添加空格。

```
package main
```

```
import (  
    "encoding/json"  
    "fmt"  
)  
  
type Student struct {  
    ID      int    `json:"id"` //通过指定 tag 实现 json 序列化该字段时的 key  
    Gender string `json:"gender"`  
    Name     string  
    Sno      string  
}  
  
func main() {  
    var s1 = Student{  
        ID:      1,  
        Gender: "男",  
        Name:    "李四",  
        Sno:     "s0001",  
    }  
    fmt.Printf("%#v\n", s1)  
    var s, _ = json.Marshal(s1)  
    jsonStr := string(s)  
    fmt.Println(jsonStr)  
}
```

```
package main  
  
import (  
    "encoding/json"
```

```
"fmt"
)

type Student struct {

    ID      int    `json:"id"` //通过指定 tag 实现 json 序列化该字段时的 key

    Gender string `json:"gender"`

    Name     string

    Sno      string

}

func main() {

    var s2 Student

    var str = "{\"id\":1,\"gender\":\"男\",\"Name\":\"李四\",\"Sno\":\"s0001\"}"

    err := json.Unmarshal([]byte(str), &s2)

    if err != nil {

        fmt.Println(err)

    }

    fmt.Printf("%#v", s2)

}
```

四、嵌套结构体和 JSON 序列化反序列化

```
package main

import (

    "encoding/json"

    "fmt"
```



```
)

//Student 学生
type Student struct {
    ID      int
    Gender  string
    Name     string
}

//Class 班级
type Class struct {
    Title      string
    Students []Student
}

func main() {
    c := &Class{
        Title:      "001",
        Students: make([]Student, 0, 200),
    }

    for i := 0; i < 10; i++ {
        stu := Student{
            Name:      fmt.Sprintf("stu%02d", i),
            Gender: "男",
            ID:         i,
        }

        c.Students = append(c.Students, stu)
    }
}
```

```
}

//JSON 序列化: 结构体-->JSON 格式的字符串

data, err := json.Marshal(c)

if err != nil {

    fmt.Println("json marshal failed")

    return

}

fmt.Printf("json:%s\n", data)

}
```

```
package main

import (

    "encoding/json"

    "fmt"

)

//Student 学生

type Student struct {

    ID      int

    Gender  string

    Name    string

}

//Class 班级

type Class struct {

    Title    string

    Students []Student

}

func main() {
```



```

str := `{"Title":"001","Students":[{"ID":0,"Gender":"男","Name":"stu00"},{"ID":1,"Gender":"男","Name":"stu01"},{"ID":2,"Gender":"男","Name":"stu02"},{"ID":3,"Gender":"男","Name":"stu03"},{"ID":4,"Gender":"男","Name":"stu04"},{"ID":5,"Gender":"男","Name":"stu05"},{"ID":6,"Gender":"男","Name":"stu06"},{"ID":7,"Gender":"男","Name":"stu07"},{"ID":8,"Gender":"男","Name":"stu08"},{"ID":9,"Gender":"男","Name":"stu09"}]}`

c1 := &Class{}

err := json.Unmarshal([]byte(str), c1)

if err != nil {

    fmt.Println("json unmarshal failed!")

    return

}

fmt.Printf("%#v\n", c1)
}

```

五、关于 Map、切片的序列化反序列化

Map 和切片也可以进行序列化和反序列化，这个我们讲完接口后再去给大家详细讲解