Go 语言的类型转换

Go语言是强类型语言，各种类型之间无法做强制类型转换。而接口对接开发时处理的数据本身类型充满不确定性，需要做一系列兼容保证程序编译通过，尽量避免出现panic，保证程序正常执行。本文将以开发一个“restful 接口”的实际业务需求为例，介绍Go语言中因将json解码为无类型数据引起的类型转换问题，涉及:

- json处理
- 理解nil
- 无类型是什么类型
- 什么时候要做类型断言
- 常见的数字字符串间转换
- 反射在类型判断中的应用

一、业务场景分析

在一次业务接口对接需求的开发中，我们通过http client调用后端接口得到后端返回的数据是json格式的，形如：

```
{
    // 业务状态码
     code:200,
     // 提示信息
     message:"OK",
     // 具体数据
     data:{
         name:"张三"
     }
}
```

但数据中的各个字段类型不确定：

- code: 200 或 “200”
- message: “error Message” 或 null
- data: 空数组、对象、字符串 或 null

- data.name: 不存在该索引、字符串 或 null

来看看我们的开发任务：

- 解析json
- code 等于 200 时拿到data.name并返回{code:200,data:jsonData.data.name}
- code 不等于 200 时拿到Message并添加一段我们的报错描述

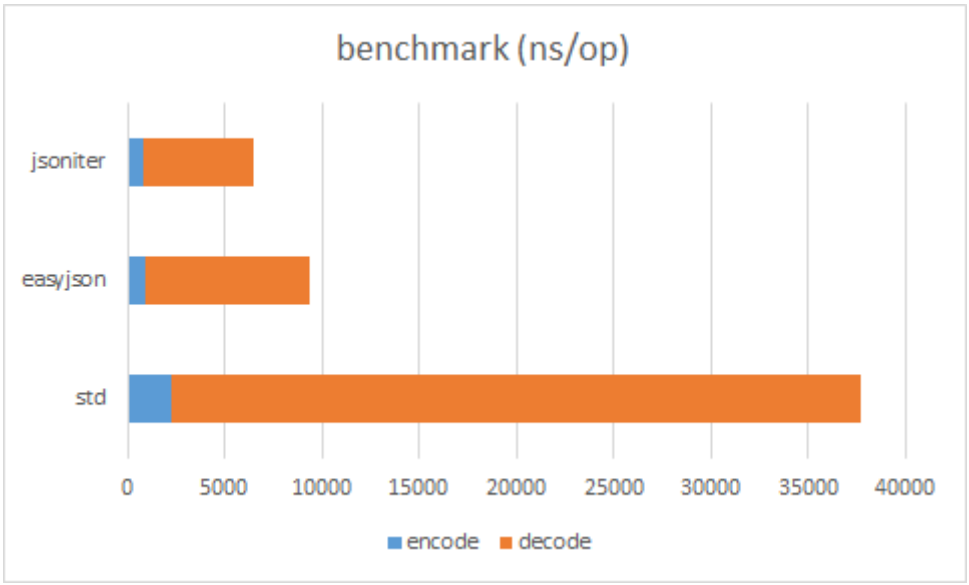
二、开发实践

解析JSON

- json内部结构不确定
- json内部字段的类型不确定

拿到这样的一个json字符串并解析为Go语言内部的结构体有三种方案：

- go标准包 `encoding/json`
- bitly公司开源的go-simplejson `github.com/bitly/go-simplejson`
- jsoniter `github.com/json-iterator/go`



jsoniter提供了与go标准包兼容的方法以及更多的方法来简化对json的操作，还具有优秀的性能，对于日常开发可以采用。

```
var dat map[string] interface{}
err := jsoniter.Unmarshal(jsonStr, &dat)
```

通过上面的代码我们拿到的dat变量是一个map，内部成员变量的值的类型都是interface{}，也可以认为是不确定的类型，需要后续依次处理。

判断数据中code等于200

- 条件判断语句中条件表达式返回值必须为bool类型
- == 相等判断成立的条件是左右表达式的返回值一致且类型一致

表达式 `dat["code"] == 200` 报错，因为我们 `json.Unmarshal` 解出的数据定义的类型是 `map[string]interface{}` ，也就是dat里每个索引的类型是interface{}

试试强制转换：

```
int(dat["code"])
```

无法转换，需要类型断言 `cannot convert a (type interface {}) to type int: need type assertion`

那再试试类型断言：

```
dat["code"].(int)
```


编译通过了，但运行时报错。阅读文档得知 `json.Unmarshal` 转换规则。

> bool, for JSON booleans
>
> float64, for JSON numbers
>
> string, for JSON strings
>
> []interface{}, for JSON arrays
>
> map[string]interface{}, for JSON objects
>
> nil for JSON null

原来JSON标准中没有浮点型或整型的区别，只有数字一种类型，而 `encoding/json` 包对于未限定返回值类型的Unmarshal会做为float64返回，所以只能断言成float64。

先类型断言为float64，再强制转换为int:

```
code := int(dat["code"].(float64))
if  code == 200 {
```

```
    ...
}
```

看起来解决了数据类型问题，但是如果dat[“code”]是字符串，上面的代码运行时类型断言就报错了，需要改成**安全的类型断言**。

```
code, err := int(dat["code"].(float64))
if  err {
    fmt.Println("type assertion error")
}
if  code == 200 {
    ...
}
```

或者采用**switch测试断言法**。

```
var code interface{}
switch value := dat["code"].(type) {
case string:
    fmt.Println("string to int")
    code, _ = strconv.Atoi(value)
case float64:
    fmt.Println("float64 to int")
    code = int(value)
case int:
    code = value
default:
    fmt.Println("unknown type")
}

fmt.Println(code)
```


处理可能为null的字符串

- Go中字符串不能赋值nil
- string类型不可改变
- string类型与[]byte类型可以方便的相互转换

现在我们取值dat[“data”][“name”]

但dat[“data”][“name”]一般情况为字符串，有时又可能是null，而Go语言中string类型是不能是null的，怎么办呢？

```
string(nil)
// cannot convert nil to type string
```

解决方案 先转成[]byte,再转成string

```
b := string([]byte(nil))
```

需要注意nil是有类型的值，我们在下文会有介绍。

字符串转为错误类型

有的时候需要把获取的字符串当成错误返回在其他地方使用。

解决方案 使用 `errors.New` 和 `fmt.Errorf` 。

```
var str string = "Error Message"
errors.New(str)

const name, id = "bimmler", 17
err := fmt.Errorf("user %q (id %d) not found", name, id)
if err != nil {
    fmt.Print(err)
}
```


三、实操总结

这个例子是实际开发中比较经典的场景，虽然业务场景比较简单，但仅转换类型一项就已经把自己搞晕了。

还好我们终于搞定了。

1. json字符串与json结构体间的转换。
2. 整数与字符串之间的转换。
3. 无类型的数字转为整数。
4. null转字符串。
5. 字符串与错误之间的转换。

四、深入学习

实际开发中一步一坑的走过来是学习的最快途径，但不够系统，为了未来开发中能够快速解决类似问题，我归纳几个问题，大家先试着答一下，后面有整理总结。

1. Go有哪些类型？零值分别是什么？在一些报错中出现的interface类型、non-interface类型、untyped无类型都是什么意思？
2. 哪些类型可以赋值nil？nil有哪些类型？
3. 哪些类型间可以进行强制转换？
4. 什么情况可以做类型断言？
5. 怎么做字符串与数字类型间的转换？
6. 除了类型转换还有什么招？

Go有哪些类型？零值分别是什么？在一些报错中出现的interface类型、non-interface类型、untyped无类型都是什么意思？

可以简单将类型分类为：

-	说明	类型举例	初始值、零值
无类型	只有常量可以是无类型	iota是一个无类型的int nil是一个无类型的nil true、false是无类型的bool	
基本类型	包含布尔、数字、字符串在内的18种类型	bool、int、float64、complex128、string	bool: false int: 0 string: “”
高级类型	用于自定义数据类型的制作工具类型	array、slice、map、func、interface、struct	相应类型的nil
自定义类型	基于基本类型和高级类型定义的类型	error是基于interface类型定义的类型	相应类型的基础类型的零值

- 根据表格可知interface类型是高级类型的一种。
- 除interface类型和基于interface类型定义的类型之外的类型统称non-interface类型。
- Go中只有常量可以是无类型的，主要是为了可以让编译器为这些常量提供比基础类型更高精度的运算。

- 当无类型常量被赋值给变量时或语句右边有含明确类型的值且转换合法的话会被隐式转换。

无类型的常量类型有：

-	默认隐式转换类型
无类型的布尔型	bool
无类型的整数	int
无类型的字符	rune
无类型的浮点数	float64
无类型的复数	complex128
无类型的字符串	string
无类型的nil	-

nil可以赋值给哪些类型？nil有哪些类型？

在golang中，nil只能赋值给指针、channel、func、interface、map或slice类型的变量。如果未遵循这个规则，则会引发panic。

换句话说nil无法赋值给布尔、数字、字符串等基本类型而可以赋值给高级类型及基于高级类型的自定义类型。

- `nil` 实际上是一个有类型的值，而未指定类型的 `nil` 的值和类型都是 `<nil>` ，这个 `<nil>` 类型也满足interface{}类型， `nil == interface{}(nil)` 表达式结果为 `true` 。
- 不同类型的 `nil` 不相等，无类型的 `nil` 与有类型的 `nil` 也不相等。
- `nil` 也是 `channels` , `slices` , `maps` , `interfaces` , `function` , `pointer` 这些类型的空值。
- `nil == interface{}((func())(nil))` 表达式结果为 `false` ，因为右边实际上还是 `func` 类型的 `nil` 值。

-	type
nil	interface
(func())(nil)	Func
interface{}(nil)	interface

-	type
map[string]string(nil)	map
[]string(nil)	Slice
(chan struct{})(nil)	Channel
(*struct{})(nil)	Pointer
(*int)(nil)	Pointer

Go语言中的nil是个很让人困扰的值，理解不了就先死记硬背吧。

-	result
nil == interface{}(nil)	true
nil == interface{}((func())(nil))	false
NIL := (func())(nil); nil == interface{}(NIL) || reflect.ValueOf(NIL).IsNil()	true

哪些类型间可以进行强制转换？

下表中为从左列类型到标题行类型的强制转换。

-	bool	int	float64	string	interface{}	error
bool	-	no	no	no	yes	no
int	no	-	yes	yes	yes	no
float64	no	yes	-	no	yes	no
string	no	no	no	-	yes	no
interface{}	no	no	no	no	-	no
error	no	no	no	no	yes	-
nil	no	no	no	no	yes	yes

1. string(97) == “a” int转string得到的是对应ASCII的字符。
2. float64转int数据被截断，实际上小类型向大类型转换时，通常都是安全的，而大类型向小类型转换会进行截断且会遇上在内存中的存储方式分为两种：大端序和小端序，这里不展开了。
3. 空接口interface{}向其他类型转换前需要类型断言。

什么情况可以做类型断言？

下表中为从左列类型到标题行类型的类型断言。

-	bool	int	float64	string	interface	error
bool	no	no	no	no	no	no
int	no	no	no	no	no	no
float64	no	no	no	no	no	no
string	no	no	no	no	no	no
interface	yes	yes	yes	yes	yes	yes
error	no	no	no	no	yes	yes

1. 只有接口类型可以进行类型断言。
2. 空interface的值的类型跟断言的类型一致时才能成功。

怎么做字符串与数字类型间的转换？

下表string解析为布尔、浮点、整形方法。

-	方法
string -> int	strconv.Atoi(“-42”)
string <- int	strconv.Itoa(-42)
string -> bool	strconv.ParseBool(“true”)
string <- bool	strconv.FormatBool(true)
string -> float64	strconv.ParseFloat(“3.1415”, 64)
string <- float64	strconv.FormatFloat(3.1415, ‘E’, -1, 64)
string -> int	strconv.ParseInt(“-42”, 10, 64)
string <- int	strconv.FormatInt(-42, 16)
string -> uint	strconv.ParseUint(“42”, 10, 64)
string <- uint	strconv. FormatUint(-42, 16)

注意字符串中如果包含非数字部分会导致转换为数字类型失败。

除了类型转换还有什么招？

Go语言中类型间转换限制这么多，有没有一种办法能像弱类型语言那样做类型检测，自动帮我们处理好数据的类型，让程序对于常见的整形、布尔、字符串能够自动转换，程序中减少一些不必要的判断呢？我们来了解下Go中的反射reflect。

reflect包本身有点难以理解，我们抛开难理解的部分直接说好理解的。

- `reflect.TypeOf(x)` 可以帮我们检测数据类型。
- `reflect.ValueOf` 返回的数据提供了众多方法可以做类型转换。
- `reflect.ValueOf(x).Kind()` 返回的数据的基础类型。
- `reflect.ValueOf(x).Type()` 返回的数据的当前类型。

看一下reflect.TypeOf的定义 `func TypeOf(i interface{}) Type`

由于所有类型的数据都满足空接口 `interface{}` ，所以任何数据都可以传入 `reflect.TypeOf` ，而返回值则是检测出的类型。


```
package main
import (
    "fmt"
    "reflect"
)

type aNewType int

func main() {
    var x1 aNewType = 1
    var x2 int = 1
    fmt.Println(reflect.TypeOf(x1))
    fmt.Println(reflect.ValueOf(x1).Type())
    //以上输出两次 main.aNewType

    fmt.Println(reflect.ValueOf(x1).Kind())
    //以上输出 int

}
```

当我们利用反射得到数据的实际类型后再进行相应的条件判断，可以有效的解决上面对未知类型的数据的类型断言和类型转换报错的尴尬。

最终我们的业务处理代码如下：


```
package main

import (
    "fmt"
    "reflect"
    "strconv"
    "encoding/json"
)

func main() {

    jsonData := []byte(`{"code":"200","message": "OK", "data":
{"name": "张三"}}`)

    var (
        code interface{}
        message interface{}
        name interface{}
    )
    type HttpData struct {
        Code    int `json:"code"`
        Message string `json:"message"`
        Data    string `json:"data"`
    }
    var dat map[string]interface{}
    err := json.Unmarshal(jsonData, &dat)
```

```
    if (err != nil) {
        fmt.Println(err)
    }
    fmt.Println(dat)
    switch reflect.ValueOf(dat["code"]).Kind() {
    case reflect.String:
        code, _ = strconv.Atoi(dat["code"].(string))
    case reflect.Float64:
        code = int(dat["code"].(float64))
    case reflect.Int:
        code = dat["code"]
    default:
        fmt.Println(reflect.ValueOf(dat["code"]).Kind())
        code = int(dat["code"].(float64))
    }

    if (reflect.ValueOf(dat["message"]).Kind() == reflect.String)
{

        message = dat["message"]
    } else {
        message = ""
    }
    if (reflect.ValueOf(dat["data"]).Kind() == reflect.Map) {

        name = (dat["data"].(map[string]interface{}))["name"]
        if (reflect.ValueOf(name).Kind() != reflect.String) {
            name = ""
        }
    } else {
        name = ""
    }

    reStruct := &HttpData{
        Code:code.(int),
        Message:message.(string),
        Data:name.(string),
    }
    result, _ := json.Marshal(reStruct)
    fmt.Println(string(result))
}
```


另外想要深入了解reflect的同学可以阅读如下链接：

1. reflect官方文档
2. Go的反射法则

第二篇是我翻译的官方博客中的一篇文章，欢迎加星。

参考资料

1. go语言基础示例
2. go语言中转换字符串到整形
3. go官方文档
4. go官方博客
5. golang: 详解interface和nil
6. Go：保留关键字及基本数据类型
7. Go中error类型的nil值和nil
8. 谈一谈Go的interface和reflect
9. Go中接口转换为整形
10. 深入go语言的nil值
11. 50种go语言中常见陷阱