链滴

Golang 小技巧——不定时更新

作者：InkDP
原文链接：https://ld246.com/article/1592573702556
来源网站：链滴

1、函数返回值定义

一般的函数定义都是：

```
func Test(a,b int) (int, int){}
```

然而go却可以这样：

```
func Test(a, b int) (c,d int) {}
```

你可能觉得没什么，但是对于我这种懒人来说，这东西可太方便了，因为go没有 try...catch，所以所的错误都需要自己手动抛出，一个函数里你可能有N个↓

```
if err != nil {
    return err
}
```

实际中，你绝对不会只返回一个 err，可能还夹杂着各种乱七八糟的东西，写一次还好，写多了你真不会烦吗？然而有了第二种定义方式，不过你又多少个返回值，只需要一个 return即可搞定，


```
func (b buriedPoint) Retention() (channel, projectId, startTime, endTime string, list []dbmodel.
uriedPointKey, data []map[string]string, err error) {
    //...
    //return channel, projectId, startTime, endTime, list, data, err
```

```
    return //选哪个不是一目了然吗，当然实际中不会让你返回这么多，这里有些夸张
}
```

当然上述的方法虽爽，但是也还是会有问题的，让我们再


```
func StringToInt(str string) (v int, err error) {
    if true {
        int, err := strconv.Atoi(str)
        if err != nil {
            return
        }
        v = int
    }
    return
}
```

乍一看没啥毛病，但是你运行下看看报不报错就完事了

```
./main.go:17:4: err is shadowed during return
```

出现问题就要去解决，提供两种方法↓

方法1：

```
func StringToInt(str string) (v int, err error) {
    if true {
        int, rErr := strconv.Atoi(str)
        if err != nil {
            err = rErr
            return
        }
        v = int
    }
    return
}
```

方法2：

```
func StringToInt(str string) (v int, err error) {
    if true {
        int, err := strconv.Atoi(str)
        if err != nil {
            return v, err
        }
        v = int
    }
    return
```

```
}
```

到这个时候，它是不是就不这么香了，是否预定义需要根据实际场景决定。

2、JSON数组返回NULL

当你的接口返回一个数组，而且数组正好为空时↓

```
{
    "code":200,
    "msg":"",
    "data":null,
}
```

你可能会返回这样的东西，那么你的前端看了可能会打人(我帮你们问过了)，去翻了下go官方的json，发现了以下内容：

Array and slice values encode as JSON arrays, except that []byte encodes as a base64-encode string, and a nil slice encodes as the null JSON value.

借助翻译软件：

数组和切片值编码为JSON数组，但[] byte编码为base64编码的字符串，而nil slice编码为Null JSO值

日常定义数组时，我们一般采用如下两种方式初始化：

```
var t []int

t := []int{}
```

函数返回值定义中定义的与上述两周并无差异，所以也就会返回一样的

定时数组时使用 make就可以完全避免这种情况

```
var t = make([]int, 0)
```

demo：

```
func main() {
    data := Str{}
    data2 := Str{Array: []string{}}
    var arr = []string{}
    data.Array = arr
    buf, err := json.Marshal(&data)
    log.Println(string(buf), err)
    buf2, err2 := json.Marshal(&data2)
    log.Println(string(buf2), err2)
}
```