Go语言结构体标签（Struct Tag）

通过 reflect.Type 获取结构体成员信息 reflect.StructField 结构中的 Tag 被称为结构体标签（Struct Tag）。结构体标签是对结构体字段的额外信息标签。

JSON、BSON 等格式进行序列化及对象关系映射（Object Relational Mapping，简称 ORM）系统都会用到结构体标签，这些系统使用标签设定字段在处理时应该具备的特殊属性和可能发生的行为。这些信息都是静态的，无须实例化结构体，可以通过反射获取到。

提示

结构体标签（Struct Tag）类似于 C# 中的特性（Attribute）。C# 允许在类、字段、方法等前面添加 Attribute，然后在反射系统中可以获取到这个属性系统。例如：

```
[Conditional("DEBUG")]
public static void Message(string msg)
{
    Console.WriteLine(msg);
}
```

结构体标签的格式

Tag 在结构体字段后方书写的格式如下：

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

结构体标签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。键值对之间使用一个空格分隔。

从结构体标签中获取值

StructTag 拥有一些方法，可以进行 Tag 信息的解析和提取，如下所示：

- func(tag StructTag)Get(key string)string
- 根据 Tag 中的键获取对应的值，例如 `key1:"value1"key2:"value2"` 的 Tag 中，可以传入“key1”获得“value1”。
- func(tag StructTag)Lookup(key string)(value string,ok bool)
- 根据 Tag 中的键，查询值是否存在。

结构体标签格式错误导致的问题

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，参见下面这个例子：

```
01. package main
```

```
02.
03. import (
04.     "fmt"
05.     "reflect"
06. )
07.
08. func main() {
09.
10.     type cat struct {
11.         Name string
12.         Type int `json: "type" id:"100"`
13.     }
14.
15.     typeOfCat := reflect.TypeOf(cat{})
16.
17.     if catType, ok := typeOfCat.FieldByName("Type"); ok {
18.
19.         fmt.Println(catType.Tag.Get("json"))
20.     }
21.
22. }
```

代码输出空字符串，并不会输出期望的 type。

第 12 行中，在 `json:` 和 `"type"` 之间增加了一个空格。这种写法没有遵守结构体标签的规则，因此无法通过 Tag.Get 获取到正确的 json 对应的值。

这个错误在开发中非常容易被疏忽，造成难以察觉的错误。