json序列化

json 序列化是指，将有 key-value 结构的数据类型(比如结构体、map、切片)序列化成 json 字符串的操作。

- 结构体序列化
- map序列化
- 切片序列化

结构体序列化

对于结构体的序列化，如果我们希望序列化后的 key 的名字，又我们自己重新制定，那么可以给 struct指定一个 tag 标签.

反序列化

在反序列化一个json字符串时，要确保反序列化后的数据类型和原来序列化前的数据类型一致。 //如果 json 字符串是通过程序获取到的，则不需要再对 “ 转义处理。

JSON转切片

```go
package main

import (
    "encoding/json"
    "fmt"
)

func main() {
    // 定义JSON格式的字符串
    data := `[{"Level": "debug", "Msg": "xxxxxx"}]`
    var dbgInfos []map[string] string
    // 将字符串解析成map切片
    json.Unmarshal([] byte(data), &dbgInfos)
    fmt.Println(dbgInfos)
}
```

运行结果

```
[Running] go run "[redacted]/go_notes/base/go_json_to_map.go"
[map[Level:debug Msg:xxxxxx]]

[Done] exited with code=0 in 0.185 seconds
```

在解码过程中，json包会将JSON类型转换为Go类型，转换规则如下:

> JSON boolean -> bool
> JSON number -> float64 JSON string -> string JSON 数组 -> [] interface{} JSON
>
> object -> map null -> nil

JSON 转结构体

结构体字段标签

解码时依然支持结构体字段标签，规则和编码时一样

```go
package main
import (
  "encoding/json"
  "fmt"
)

type DebugInfo struct {
  Level string `json: "level"` // level 解码为Level
  Msg  string `json: "messgage"` // message 解码为Msg
  Author string `json: "-"`         //   忽略Author
}

func (dbgInfo DebugInfo) String() string {
  return fmt.Sprintf("{Level: %v, Msg: %v, Author: %v}", dbgInfo.Level, dbgInfo.Msg, dbgInfo.Author)
}

func main () {
  // 定义json格式字符串
  data := `[{"level": "debug", "message": "File Not Found", "author": "AlexZ"},` + `{"level": "", "message": "Logic error", "author": "Tom"}]`
  var dbgInfos []DebugInfo
  // 将字符串按照标签解析成结构体切片
  json.Unmarshal([] byte(data), &dbgInfos)
  fmt.Println(dbgInfos)
}
```

```
[Running] go run "[redacted]/go_notes/base/go_struct_field_label.go"
[{Level: debug, Msg: , Author: AlexZ} {Level: , Msg: , Author: Tom}]

[Done] exited with code=0 in 0.152 seconds
```