2017年06月21日 • GO by taowen

[转]Golang 中使用 JSON 的小技巧

目录 [-]

taowen是json-iterator的作者。 序列化和反序列化需要处理JSON和struct的关系，其中会用到一些技巧。 原文Golang 中使用 JSON 的小技巧是他的经验之谈，介绍了一些struct解析成json的技巧，以及 json-iterator 库的一些便利的处理。

有的时候上游传过来的字段是string类型的，但是我们却想用变成数字来使用。 本来用一个json:",string" 就可以支持了，如果不知道golang的这些小技巧，就要大费周章了。

参考文章：http://attilaolah.eu/2014/09/10/json-and-struct-composition-in-go/

1 临时忽略struct空字段

```
type User struct {
    Email    string `json:"email"`
    Password string `json:"password"`
    // many more fields…
}
```

如果想临时忽略掉空`Password`字段,可以用`omitempty`:

```
json.Marshal(struct {
    *User
    Password bool `json:"password,omitempty"`
}{
    User: user,
})
```

2 临时添加额外的字段

```
type User struct {
    Email    string `json:"email"`
    Password string `json:"password"`
    // many more fields…
}
```

临时忽略掉空`Password`字段，并且添加`token`字段

```
json.Marshal(struct {
    *User
    Token    string `json:"token"`
    Password bool `json:"password,omitempty"`
}{
    User: user,
    Token: token,
})
```

3 临时粘合两个struct

通过嵌入struct的方式:

```
type BlogPost struct {
    URL   string `json:"url"`
    Title string `json:"title"`
}

type Analytics struct {
    Visitors  int `json:"visitors"`
    PageViews int `json:"page_views"`
}

json.Marshal(struct{
    *BlogPost
    *Analytics
}{post, analytics})
```

4 一个json切分成两个struct

```
json.Unmarshal([]byte(`{
  "url": "attila@attilaolah.eu",
  "title": "Attila's Blog",
  "visitors": 6,
  "page_views": 14
}`), &struct {
  *BlogPost
  *Analytics
}{&post, &analytics})
```

5 临时改名struct的字段

```
type CacheItem struct {
    Key    string `json:"key"`
    MaxAge int    `json:"cacheAge"`
    Value  Value  `json:"cacheValue"`
}

json.Marshal(struct{
    *CacheItem

    // Omit bad keys
    OmitMaxAge omit `json:"cacheAge,omitempty"`
    OmitValue  omit `json:"cacheValue,omitempty"`

```

```
    // Add nice keys
    MaxAge int    `json:"max_age"`
    Value  *Value `json:"value"`
}{
    CacheItem: item,

    // Set the int by value:
    MaxAge: item.MaxAge,

    // Set the nested struct by reference, avoid making a copy:
    Value: &item.Value,
})
```

6 用字符串传递数字

```
type TestObject struct {
    Field1 int    `json:",string"`
}
```

这个对应的json是 `{"Field1": "100"}`

如果json是 `{"Field1": 100}` 则会报错

7 容忍字符串和数字互转

如果你使用的是jsoniter，可以启动 **模糊模式** 来支持 PHP 传递过来的 JSON。

```
import "github.com/json-iterator/go/extra"

extra.RegisterFuzzyDecoders()
```

这样就可以处理字符串和数字类型不对的问题了。比如

```
var val string
jsoniter.UnmarshalFromString(`100`, &val)
```

又比如

```
var val float32
jsoniter.UnmarshalFromString(`"1.23"`, &val)
```

8 容忍空数组作为对象

PHP另外一个令人崩溃的地方是，如果 PHP array是空的时候，序列化出来是`[]`。但是不为空的时候，序列化出来的是`{"key":"value"}`。 我们需要把 `[]` 当成 `{}` 处理。

如果你使用的是jsoniter，可以启动模糊模式来支持 PHP 传递过来的 JSON。

```
import "github.com/json-iterator/go/extra"

extra.RegisterFuzzyDecoders()
```

这样就可以支持了

```
var val map[string]interface{}
jsoniter.UnmarshalFromString(`[]`, &val)
```

9 使用 MarshalJSON支持time.Time

golang 默认会把 `time.Time` 用字符串方式序列化。如果我们想用其他方式表示 `time.Time`，需要自定义类型并定义 `MarshalJSON`。

```
type timeImplementedMarshaler time.Time

func (obj timeImplementedMarshaler) MarshalJSON() ([]byte, error) {
    seconds := time.Time(obj).Unix()
    return []byte(strconv.FormatInt(seconds, 10)), nil
}
```

序列化的时候会调用 MarshalJSON

```
type TestObject struct {
    Field timeImplementedMarshaler
}
should := require.New(t)
val := timeImplementedMarshaler(time.Unix(123, 0))
obj := TestObject{val}
bytes, err := jsoniter.Marshal(obj)
should.Nil(err)
should.Equal(`{"Field":123}`, string(bytes))
```

10 使用 RegisterTypeEncoder支持time.Time

jsoniter 能够对不是你定义的type自定义JSON编解码方式。比如对于 `time.Time` 可以用 epoch int64 来序列化

```
import "github.com/json-iterator/go/extra"

extra.RegisterTimeAsInt64Codec(time.Microsecond)
output, err := jsoniter.Marshal(time.Unix(1, 1002))
should.Equal("1000001", string(output))
```

如果要自定义的话，参见 `RegisterTimeAsInt64Codec` 的实现代码

11 使用 MarshalText支持非字符串作为key的map

虽然 JSON 标准里只支持 `string` 作为 `key` 的 `map`。但是 golang 通过 `MarshalText()` 接口，使得其他类型也可以作为 `map` 的 `key`。例如

```
f, _, _ := big.ParseFloat("1", 10, 64, big.ToZero)
val := map[*big.Float]string{f: "2"}
str, err := MarshalToString(val)
should.Equal(`{"1":"2"}`, str)
```

其中 `big.Float` 就实现了 `MarshalText()`

12 使用 json.RawMessage

如果部分json文档没有标准格式，我们可以把原始的信息用`[]byte`保存下来。

```
type TestObject struct {
    Field1 string
    Field2 json.RawMessage
}
var data TestObject
json.Unmarshal([]byte(`{"field1": "hello", "field2": [1,2,3]}`), &data)
should.Equal(` [1,2,3]`, string(data.Field2))
```

13 使用 json.Number

默认情况下，如果是 `interface{}` 对应数字的情况会是 `float64` 类型的。如果输入的数字比较大，这个表示会有损精度。所以可以 `UseNumber()` 启用 `json.Number` 来用字符串表示数字。

```
decoder1 := json.NewDecoder(bytes.NewBufferString(`123`))
decoder1.UseNumber()
var obj1 interface{}
decoder1.Decode(&obj1)
should.Equal(json.Number("123"), obj1)
```

jsoniter 支持标准库的这个用法。同时，扩展了行为使得 `Unmarshal` 也可以支持 `UseNumber` 了。

```
json := Config{UseNumber:true}.Froze()
var obj interface{}
json.UnmarshalFromString("123", &obj)
should.Equal(json.Number("123"), obj)
```

14 统一更改字段的命名风格

经常 JSON 里的字段名 Go 里的字段名是不一样的。我们可以用 field tag 来修改。

```
output, err := jsoniter.Marshal(struct {
    UserName      string `json:"user_name"`
    FirstLanguage string `json:"first_language"`
}{
    UserName:      "taowen",
    FirstLanguage: "Chinese",
})
should.Equal(`{"user_name":"taowen","first_language":"Chinese"}`, string(outp
```

但是一个个字段来设置，太麻烦了。如果使用 jsoniter，我们可以统一设置命名风格。

```
import "github.com/json-iterator/go/extra"

extra.SetNamingStrategy(LowerCaseWithUnderscores)
output, err := jsoniter.Marshal(struct {
    UserName      string
    FirstLanguage string
}{
    UserName:      "taowen",
    FirstLanguage: "Chinese",
})
should.Nil(err)
should.Equal(`{"user_name":"taowen","first_language":"Chinese"}`, string(out
```

15 使用私有的字段

Go 的标准库只支持 public 的 field。jsoniter 额外支持了 private 的 field。需要使用 `SupportPrivateFields()` 来开启开关。

```
import "github.com/json-iterator/go/extra"

extra.SupportPrivateFields()
type TestObject struct {
    field1 string
}
obj := TestObject{}
jsoniter.UnmarshalFromString(`{"field1":"Hello"}`, &obj)
should.Equal("Hello", obj.field1)
```

下面是我补充的内容

16 忽略掉一些字段

原文中第一节有个错误，我更正过来了。`omitempty`不会忽略某个字段，而是忽略空的字段，当字段的值为空值的时候，它不会出现在JSON数据中。

如果想忽略某个字段，需要使用 `json:"-"`格式。

```
type User struct {
    Email    string `json:"email"`
    Password string `json:"password"`
    // many more fields…
}
```

如果想临时忽略掉空`Password`字段,可以用`-`:

```
json.Marshal(struct {
    *User
    Password bool `json:"-"`
}{
    User: user,
})
```

17 忽略掉一些字段2

如果不想更改原struct,还可以使用下面的方法：

```
type User struct {
    Email    string `json:"email"`
    Password string `json:"password"`
    // many more fields…
}

type omit *struct{}

type PublicUser struct {
    *User
    Password omit `json:"-"`
}

json.Marshal(PublicUser{
    User: user,
})
```

RXDA commented over 1 year ago

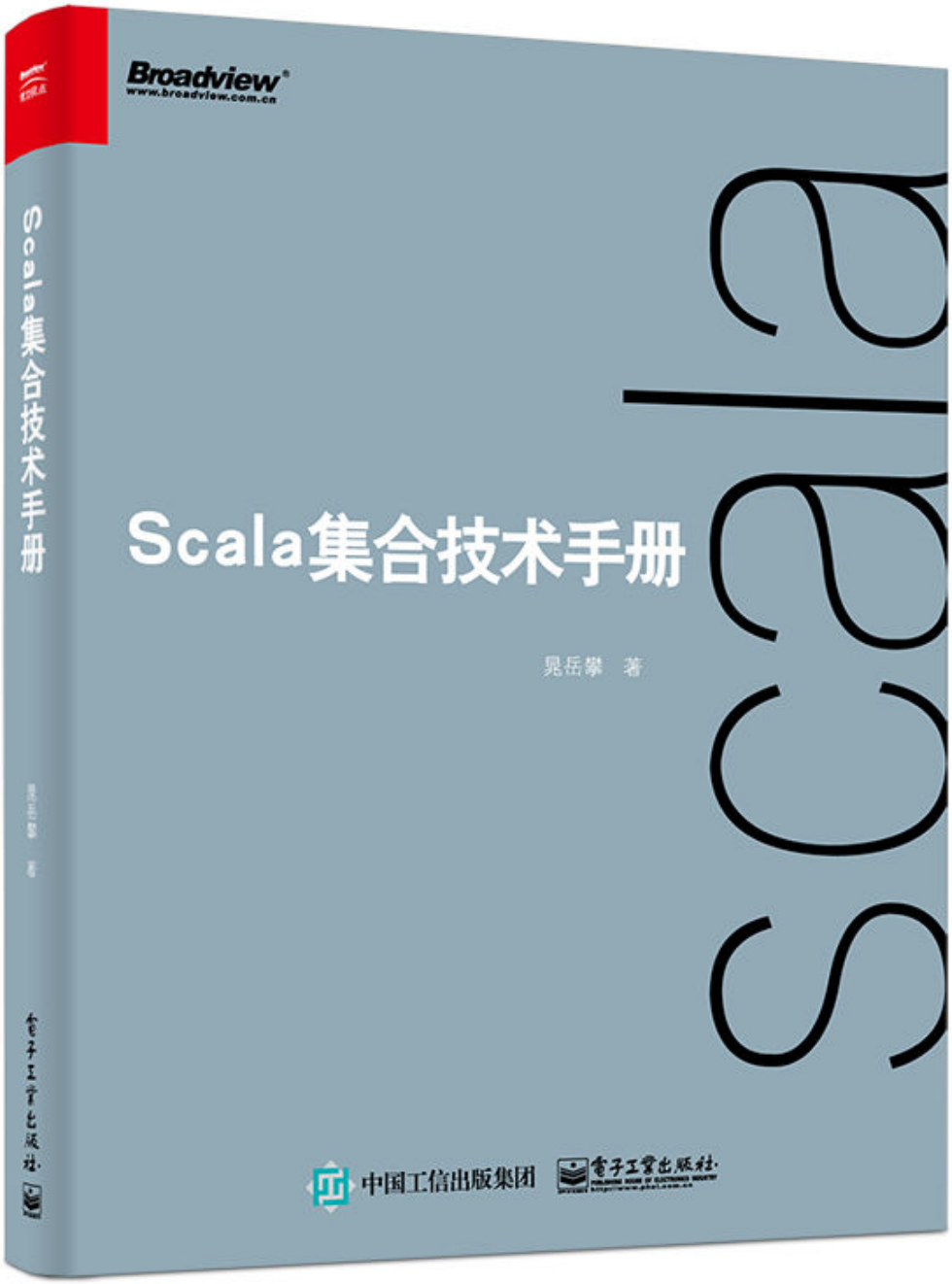
password,omitempty 才可以隐藏

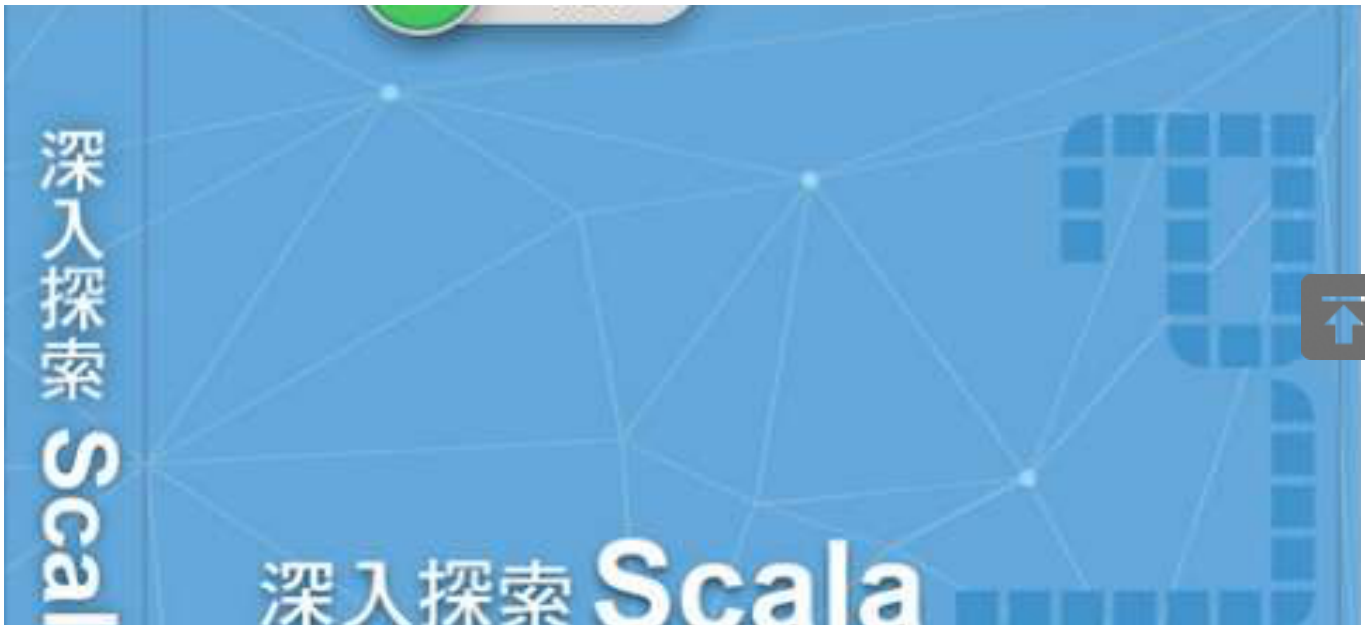
NEWER

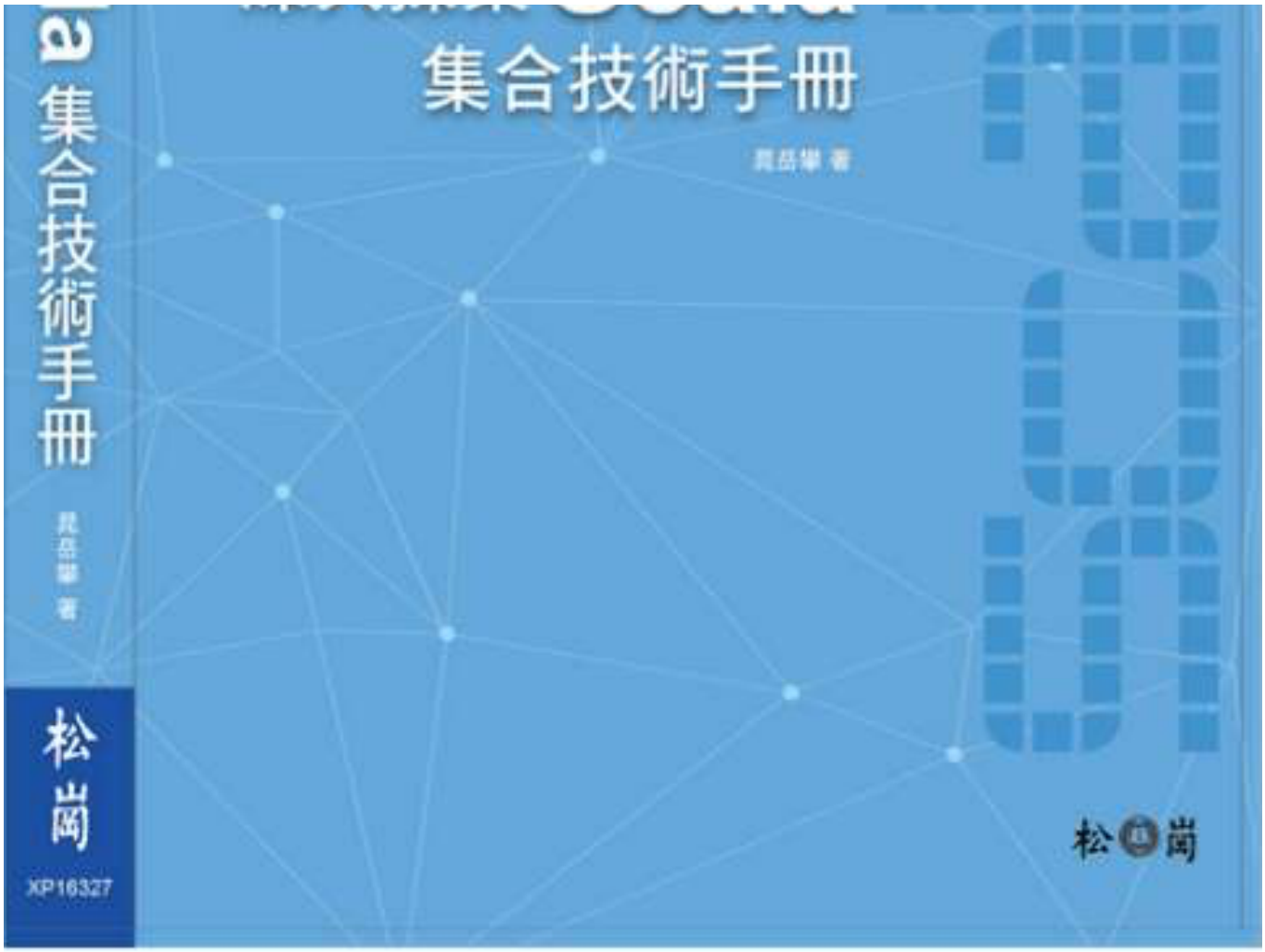
了解 Go 1.9 的类型别名

OLDER

Go 1.9 的新特性


Broadview
www.broadview.com.cn
Scala集合技术手册
晁岳攀 著
scala
中国工信出版集团
電子工業出版社


深入探索Scala



分类

高并发编程 (20)

标签云

Android ApacheBench Bower C# CDN CQRS CRC CSS CompletableFuture Comsat Curator DSL Disruptor Docker Ember FastJson Fiber GAE GC Gnuplot Go Gradle Grunt Gulp Hadoop Hazelcast IPFS Ignite JVM Java Kafka Lambda Linux LongAdder MathJax Maven Memcached Metrics Mongo Netty

归档

June 2021 (1)
May 2021 (3)
April 2021 (2)
March 2021 (2)
February 2021 (1)
January 2021 (2)
December 2020 (3)
November 2020 (2)
September 2020 (1)
August 2020 (1)
July 2020 (2)
June 2020 (2)
May 2020 (4)
April 2020 (1)
March 2020 (3)
February 2020 (2)
January 2020 (5)
December 2019 (6)
November 2019 (2)
October 2019 (6)
September 2019 (7)
August 2019 (7)
July 2019 (7)
June 2019 (1)
May 2019 (2)
April 2019 (3)
March 2019 (1)
February 2019 (6)
January 2019 (5)
December 2018 (2)
November 2018 (4)
October 2018 (2)
September 2018 (6)
August 2018 (5)
July 2018 (3)
June 2018 (3)
May 2018 (2)
April 2018 (1)
March 2018 (6)

近期文章

轻轻松松打印网页并生成pdf文档
[译]strace的10个命令

友情链接

技术栈

开发者头条
码农周刊
编程狂人周刊
importnew
并发编程网

github
stackoverflow
javacodegeeks
infoq
dzone
leetcode
jenkov
HowToDoInJava
java design patterns

Netflix技术博客
Techie Delight
Linkedin技术博客
Dropbox技术博客
Facebook技术博客
淘宝中间件团队
美团技术博客
360技术博客
小米信息部技术团队