GO-INI英文文档

书栈(BookStack.CN)

目　录

致谢

当前文档 《GO-INI英文文档》 由 进击的皇虫 使用 书栈(BookStack.CN) 进行构建，生成于 2018-07-08。

书栈(BookStack.CN) 仅提供文档编写、整理、归类等功能，以及对文档内容的生成和导出工具。

文档内容由网友们编写和整理，书栈(BookStack.CN) 难以确认文档内容知识点是否错漏。如果您在阅读文档获取知识的时候，发现文档内容有不恰当的地方，请向我们反馈，让我们共同携手，将知识准确、高效且有效地传递给每一个人。

同时，如果您在日常工作、生活和学习中遇到有价值有营养的知识文档，欢迎分享到 书栈(BookStack.CN) ，为知识的传承献上您的一份力量！

如果当前文档生成时间太久，请到 书栈(BookStack.CN) 获取最新的文档，以跟上知识更新换代的步伐。

文档地址: http://www.bookstack.cn/books/go-ini

书栈官网: http://www.bookstack.cn

书栈开源: https://github.com/TruthHun

分享，让知识传承更久远！ 感谢知识的创造者，感谢知识的分享者，也感谢每一位阅读到此处的读者，因为我们都将成为知识的传承者。

简介

INI

Package `ini` provides INI file read and write functionality in Go.

Features

- Load from multiple data sources ([]byte, file and io.ReadCloser) with overwrites.
- Read with recursion values.
- Read with parent-child sections.
- Read with auto-increment key names.
- Read with multiple-line values.
- Read with tons of helper methods.
- Read and convert values to Go types.
- Read and **WRITE** comments of sections and keys.
- Manipulate sections, keys and comments with ease.
- Keep sections and keys in order as you parse and save.

Getting Help

- API Documentation
- File An Issue

> 原文: *https://ini.unknwon.io/docs/intro*

下载安装

Installation

You must have Go installed properly on your machine with correct `$GOPATH` setup.

Download the package

To use a tagged revision:

```
$ go get gopkg.in/ini.v1
```

To use with latest changes:

```
$ go get github.com/go-ini/ini
```

Please add `-u` flag to update in the future.

Testing

If you want to test on your machine, please apply `-t` flag:

```
$ go get -t gopkg.in/ini.v1
```

Please add `-u` flag to update in the future.

Running tests on your local machine

Go to the directory of the package and execute `make test` command (**replace import path as needed**):

```
$ cd $GOPATH/src/gopkg.in/ini.v1
$ make test
go test -v -cover -race
=== RUN   Test_Version

   Get version ✔


```

```
1 total assertion

--- PASS: Test_Version (0.00s)
=== RUN   Test_isSlice

  Check if a string is in the slice ✔✔


3 total assertions

--- PASS: Test_isSlice (0.00s)
=== RUN   TestEmpty

...

--- PASS: Test_Duration (0.00s)
PASS
coverage: 94.3% of statements
ok      gopkg.in/ini.v1    1.121s
```

> 原文: *https://ini.unknwon.io/docs/intro/installation*

开始使用

Getting Started

We will go through a very simple example to illustrate how to get started.

First of all, create two files (`my.ini` and `main.go`) under the directory of your choice, let's say we choose `/tmp/ini` .

```
$ mkdir -p /tmp/ini
$ cd /tmp/ini
$ touch my.ini main.go
$ tree .
.
├── main.go
└── my.ini

0 directories, 2 files
```

Now, we put some content into the `my.ini` file (*partially take from Grafana*).

```
# possible values : production, development
app_mode = development

[paths]
# Path to where grafana can store temp files, sessions, and the sqlite3 db (if that is used)
data = /home/git/grafana

[server]
# Protocol (http or https)
protocol = http

# The http port  to use
http_port = 9999

# Redirect to correct domain if host header does not match domain
# Prevents DNS rebinding attacks
enforce_domain = true
```

Great, let’s start writing some code in `main.go` to manipulate this file.

```
1.  package main
2.
3.  import (
4.      "fmt"
5.      "os"
6.
7.      "gopkg.in/ini.v1"
8.  )
9.
10. func main() {
11.     cfg, err := ini.Load("my.ini")
12.     if err != nil {
13.         fmt.Printf("Fail to read file: %v", err)
14.         os.Exit(1)
15.     }
16.
17.     // Classic read of values, default section can be represented as empty
    string
18.     fmt.Println("App Mode:", cfg.Section("").Key("app_mode").String())
19.     fmt.Println("Data Path:", cfg.Section("paths").Key("data").String())
20.
21.     // Let's do some candidate value limitation
22.     fmt.Println("Server Protocol:",
23.         cfg.Section("server").Key("protocol").In("http", []string{"http",
    "https"}))
24.     // Value read that is not in candidates will be discarded and fall back to
    given default value
25.     fmt.Println("Email Protocol:",
26.         cfg.Section("server").Key("protocol").In("smtp", []string{"imap",
    "smtp"}))
27.
28.     // Try out auto-type conversion
29.     fmt.Printf("Port Number: (%[1]T) %[1]d\n",
    cfg.Section("server").Key("http_port").MustInt(9999))
30.     fmt.Printf("Enforce Domain: (%[1]T) %[1]v\n",
    cfg.Section("server").Key("enforce_domain").MustBool(false))
31.
32.     // Now, make some changes and save it
33.     cfg.Section("").Key("app_mode").SetValue("production")
34.     cfg.SaveTo("my.ini.local")
```

```
}
```

Almost there, let’s run this program and check the output.

```
$ go run main.go
App Mode: development
Data Path: /home/git/grafana
Server Protocol: http
Email Protocol: smtp
Port Number: (int) 9999
Enforce Domain: (bool) true

$ cat my.ini.local
# possible values : production, development
app_mode = production

[paths]
# Path to where grafana can store temp files, sessions, and the sqlite3 db (if that is used)
data = /home/git/grafana
...
```

Perfect! Though the example is very basic and covered only a small bit of all functionality, but it’s a good start.

> 原文: *https://ini.unknwon.io/docs/intro/getting_started*

我应该如何...

- 从数据源加载
- 操作分区（Section）
- 操作键（Key）
- 操作键值（Value）
- 操作注释（Comment）

从数据源加载

Load from data sources

As advertised, load from different data sources is possible and effortless.

So, what is a **Data Source** anyway?

A **Data Source** is either raw data in type `[]byte`, a file name with type `string` or `io.ReadCloser`. You can load **as many data sources as you want**. Passing other types will simply return an error.

```
cfg, err := ini.Load(
    []byte("raw data"), // Raw data
    "filename",         // File
    ioutil.NopCloser(bytes.NewReader([]byte("some other data"))),
)
```

Or start with an empty object:

```
cfg := ini.Empty()
```

When you cannot decide how many data sources to load at the beginning, you will still be able to `Append()` them later.

```
err := cfg.Append("other file", []byte("other raw data"))
```

If you have a list of files with possibilities that some of them may not available at the time, and you don't know exactly which ones, you can use `LooseLoad()` to ignore nonexistent files without returning error.

```
cfg, err := ini.LooseLoad("filename", "filename_404")
```

The cool thing is, whenever the file is available to load while you're calling `Reload()` method, it will be counted as usual.

Data Overwrite

One minor thing you need to keep in mind before you go. Data overwrite will

happen when you load from more than one data sources. The value with same key name in the later data source will overwrite whatever previously read.

For example, if you load two files `my.ini` and `my.ini.local` (example input and output from Getting Started), value of `app_mode` will be `production` not `development`.

```
cfg, err := ini.Load("my.ini", "my.ini.local")
...

cfg.Section("").Key("app_mode").String() // production
```

The only exception to data overwrite rule is when you use ShadowLoad.

Save your configuration

Finally, it’s time to save your configuration to somewhere.

A typical way to save configuration is writing it to a file:

```
// ...
err = cfg.SaveTo("my.ini")
err = cfg.SaveToIndent("my.ini", "\t")
```

Another way to save is writing to a `io.Writer` interface:

```
// ...
cfg.WriteTo(writer)
cfg.WriteToIndent(writer, "\t")
```

By default, spaces are used to align “=” sign between key and values, to disable that:

```
ini.PrettyFormat = false
```

> 原文: *https://ini.unknwon.io/docs/howto/load_data_sources*

操作分区（Section）

Working with sections

To get a section, you would need to:

```
1.  sec, err := cfg.GetSection("section name")
```

For a shortcut for default section, just give an empty string as name:

```
1.  sec, err := cfg.GetSection("")
```

Alternatively, you can use `ini.DEFAULT_SECTION` as section name to access the default section:

```
1.  sec, err := cfg.GetSection(ini.DEFAULT_SECTION)
```

When you’re pretty sure the section exists, following code could make your life easier:

```
1.  sec := cfg.Section("section name")
```

What happens when the section somehow does not exist? Don’t panic, it automatically creates and returns a new section to you.

To create a new section:

```
1.  err := cfg.NewSection("new section")
```

To get a list of sections or section names:

```
1.  secs := cfg.Sections()
2.  names := cfg.SectionStrings()
```

Parent-child Sections

You can use `.` in section name to indicate parent-child relationship between two or more sections. If the key not found in the child section,

library will try again on its parent section until there is no parent section.

```
NAME = ini
VERSION = v1
IMPORT_PATH = gopkg.in/%(NAME)s.%(VERSION)s

[package]
CLONE_URL = https://%(IMPORT_PATH)s

[package.sub]
```

```
cfg.Section("package.sub").Key("CLONE_URL").String()    //
https://gopkg.in/ini.v1
```

Unparseable Sections

Sometimes, you have sections that do not contain key-value pairs but raw content, to handle such case, you can use `LoadOptions.UnparsableSections` :

```
cfg, err := ini.LoadSources(ini.LoadOptions{
    UnparseableSections: []string{"COMMENTS"},
}, `[COMMENTS]
<1><L.Slide#2> This slide has the fuel listed in the wrong units <e.1>`)

body := cfg.Section("COMMENTS").Body()

/* --- start ---
<1><L.Slide#2> This slide has the fuel listed in the wrong units <e.1>
------  end  --- */
```

> 原文: *https://ini.unknwon.io/docs/howto/work_with_sections*

操作键（Key）

Working with keys

To get a key under a section:

```
key, err := cfg.Section("").GetKey("key name")
```

Same rule applies to key operations:

```
key := cfg.Section("").Key("key name")
```

To check if a key exists:

```
yes := cfg.Section("").HasKey("key name")
```

To create a new key:

```
err := cfg.Section("").NewKey("name", "value")
```

To get a list of keys or key names:

```
keys := cfg.Section("").Keys()
names := cfg.Section("").KeyStrings()
```

To get a clone hash of keys and corresponding values:

```
hash := cfg.Section("").KeysHash()
```

Ignore cases of key name

When you do not care about cases of section and key names, you can use `InsensitiveLoad` to force all names to be lowercased while parsing.

```
cfg, err := ini.InsensitiveLoad("filename")
//...

// sec1 and sec2 are the exactly same section object
sec1, err := cfg.GetSection("Section")
```

```
sec2, err := cfg.GetSection("SecTIOn")

// key1 and key2 are the exactly same key object
key1, err := sec1.GetKey("Key")
key2, err := sec2.GetKey("KeY")
```

MySQL-like boolean key

MySQL's configuration allows a key without value as follows:

```
[mysqld]
...
skip-host-cache
skip-name-resolve
```

By default, this is considered as missing value. But if you know you're going to deal with those cases, you can assign advanced load options:

```
cfg, err := ini.LoadSources(ini.LoadOptions{
    AllowBooleanKeys: true,
}, "my.cnf")
```

The value of those keys are always `true`, and when you save to a file, it will keep in the same foramt as you read.

To generate such keys in your program, you could use `NewBooleanKey`:

```
key, err := sec.NewBooleanKey("skip-host-cache")
```

Same Key with Multiple Values

Do you ever have a configuration file like this?

```
[remote "origin"]
url = https://github.com/Antergone/test1.git
url = https://github.com/Antergone/test2.git
fetch = +refs/heads/*:refs/remotes/origin/*
```

By default, only the last read value will be kept for the key `url`. If you want to keep all copies of value of this key, you can use ShadowLoad to achieve it:

```
cfg, err := ini.ShadowLoad(".gitconfig")
// ...

f.Section(`remote "origin"`).Key("url").String()
// Result: https://github.com/Antergone/test1.git

f.Section(`remote "origin"`).Key("url").ValueWithShadows()
// Result:  []string{
//              "https://github.com/Antergone/test1.git",
//              "https://github.com/Antergone/test2.git",
//          }
```

Auto-increment Key Names

If key name is `-` in data source, then it would be seen as special syntax for auto-increment key name start from 1, and every section is independent on counter.

```
[features]
-: Support read/write comments of keys and sections
-: Support auto-increment of key names
-: Support load multiple files to overwrite key values
```

```
cfg.Section("features").KeyStrings()    // []{"#1", "#2", "#3"}
```

Retrieve parent keys available to a child section

```
cfg.Section("package.sub").ParentKeys() // ["CLONE_URL"]
```

> 原文: *https://ini.unknwon.io/docs/howto/work_with_keys*

操作键值（Value）

Working with values

To get a string value:

```
val := cfg.Section("").Key("key name").String()
```

To validate key value on the fly:

```
val := cfg.Section("").Key("key name").Validate(func(in string) string {
    if len(in) == 0 {
        return "default"
    }
    return in
})
```

If you do not want any auto-transformation (such as recursive read) for the values, you can get raw value directly (this way you get much better performance):

```
val := cfg.Section("").Key("key name").Value()
```

To check if raw value exists:

```
yes := cfg.Section("").HasValue("test value")
```

To get value with types:

```
// For boolean values:
// true when value is: 1, t, T, TRUE, true, True, YES, yes, Yes, y, ON, on, On
// false when value is: 0, f, F, FALSE, false, False, NO, no, No, n, OFF, off, Off
v, err = cfg.Section("").Key("BOOL").Bool()
v, err = cfg.Section("").Key("FLOAT64").Float64()
v, err = cfg.Section("").Key("INT").Int()
v, err = cfg.Section("").Key("INT64").Int64()
v, err = cfg.Section("").Key("UINT").Uint()
v, err = cfg.Section("").Key("UINT64").Uint64()
```

```
v, err = cfg.Section("").Key("TIME").TimeFormat(time.RFC3339)
v, err = cfg.Section("").Key("TIME").Time() // RFC3339

v = cfg.Section("").Key("BOOL").MustBool()
v = cfg.Section("").Key("FLOAT64").MustFloat64()
v = cfg.Section("").Key("INT").MustInt()
v = cfg.Section("").Key("INT64").MustInt64()
v = cfg.Section("").Key("UINT").MustUint()
v = cfg.Section("").Key("UINT64").MustUint64()
v = cfg.Section("").Key("TIME").MustTimeFormat(time.RFC3339)
v = cfg.Section("").Key("TIME").MustTime() // RFC3339

// Methods start with Must also accept one argument for default value
// when key not found or fail to parse value to given type.
// Except method MustString, which you have to pass a default value.

v = cfg.Section("").Key("String").MustString("default")
v = cfg.Section("").Key("BOOL").MustBool(true)
v = cfg.Section("").Key("FLOAT64").MustFloat64(1.25)
v = cfg.Section("").Key("INT").MustInt(10)
v = cfg.Section("").Key("INT64").MustInt64(99)
v = cfg.Section("").Key("UINT").MustUint(3)
v = cfg.Section("").Key("UINT64").MustUint64(6)
v = cfg.Section("").Key("TIME").MustTimeFormat(time.RFC3339, time.Now())
v = cfg.Section("").Key("TIME").MustTime(time.Now()) // RFC3339
```

What if my value is three-line long?

```
[advance]
ADDRESS = """404 road,
NotFound, State, 5000
Earth"""
```

Not a problem!

```
cfg.Section("advance").Key("ADDRESS").String()

/* --- start ---
404 road,
NotFound, State, 5000
Earth
------  end  --- */
```

That’s cool, how about continuation lines?

```
[advance]
two_lines = how about \
    continuation lines?
lots_of_lines = 1 \
    2 \
    3 \
    4
```

Piece of cake!

```
cfg.Section("advance").Key("two_lines").String() // how about continuation lines?
cfg.Section("advance").Key("lots_of_lines").String() // 1 2 3 4
```

Well, I hate continuation lines, how do I disable that?

```
cfg, err := ini.LoadSources(ini.LoadOptions{
    IgnoreContinuation: true,
}, "filename")
```

Holy crap!

Note that single quotes around values will be stripped:

```
foo = "some value" // foo: some value
bar = 'some value' // bar: some value
```

Sometimes you downloaded file from Crowdin has values like the following (value is surrounded by double quotes and quotes in the value are escaped):

```
create_repo="created repository <a href=\"%s\">%s</a>"
```

How do you transform this to regular format automatically?

```
cfg, err := ini.LoadSources(ini.LoadOptions{UnescapeValueDoubleQuotes: true}, "en-US.ini"))
cfg.Section("<name of your section>").Key("create_repo").String()
// You got: created repository <a href="%s">%s</a>
```

That’s all? Hmm, no.

Helper methods of working with values

To get value with given candidates:

```
v = cfg.Section("").Key("STRING").In("default", []string{"str", "arr", "types"})
v = cfg.Section("").Key("FLOAT64").InFloat64(1.1, []float64{1.25, 2.5, 3.75})
v = cfg.Section("").Key("INT").InInt(5, []int{10, 20, 30})
v = cfg.Section("").Key("INT64").InInt64(10, []int64{10, 20, 30})
v = cfg.Section("").Key("UINT").InUint(4, []int{3, 6, 9})
v = cfg.Section("").Key("UINT64").InUint64(8, []int64{3, 6, 9})
v = cfg.Section("").Key("TIME").InTimeFormat(time.RFC3339, time.Now(), []time.Time{time1, time2, time3})
v = cfg.Section("").Key("TIME").InTime(time.Now(), []time.Time{time1, time2, time3}) // RFC3339
```

Default value will be presented if value of key is not in candidates you given, and default value does not need be one of candidates.

To validate value in a given range:

```
vals = cfg.Section("").Key("FLOAT64").RangeFloat64(0.0, 1.1, 2.2)
vals = cfg.Section("").Key("INT").RangeInt(0, 10, 20)
vals = cfg.Section("").Key("INT64").RangeInt64(0, 10, 20)
vals = cfg.Section("").Key("UINT").RangeUint(0, 3, 9)
vals = cfg.Section("").Key("UINT64").RangeUint64(0, 3, 9)
vals = cfg.Section("").Key("TIME").RangeTimeFormat(time.RFC3339, time.Now(), minTime, maxTime)
vals = cfg.Section("").Key("TIME").RangeTime(time.Now(), minTime, maxTime) // RFC3339
```

Auto-split values into a slice

To use zero value of type for invalid inputs:

```
// Input: 1.1, 2.2, 3.3, 4.4 -> [1.1 2.2 3.3 4.4]
// Input: how, 2.2, are, you -> [0.0 2.2 0.0 0.0]
vals = cfg.Section("").Key("STRINGS").Strings(",")
vals = cfg.Section("").Key("FLOAT64S").Float64s(",")
vals = cfg.Section("").Key("INTS").Ints(",")
vals = cfg.Section("").Key("INT64S").Int64s(",")
```

```
vals = cfg.Section("").Key("UINTS").Uints(",")
vals = cfg.Section("").Key("UINT64S").Uint64s(",")
vals = cfg.Section("").Key("TIMES").Times(",")
```

To exclude invalid values out of result slice:

```
// Input: 1.1, 2.2, 3.3, 4.4 -> [1.1 2.2 3.3 4.4]
// Input: how, 2.2, are, you -> [2.2]
vals = cfg.Section("").Key("FLOAT64S").ValidFloat64s(",")
vals = cfg.Section("").Key("INTS").ValidInts(",")
vals = cfg.Section("").Key("INT64S").ValidInt64s(",")
vals = cfg.Section("").Key("UINTS").ValidUints(",")
vals = cfg.Section("").Key("UINT64S").ValidUint64s(",")
vals = cfg.Section("").Key("TIMES").ValidTimes(",")
```

Or to return nothing but error when have invalid inputs:

```
// Input: 1.1, 2.2, 3.3, 4.4 -> [1.1 2.2 3.3 4.4]
// Input: how, 2.2, are, you -> error
vals = cfg.Section("").Key("FLOAT64S").StrictFloat64s(",")
vals = cfg.Section("").Key("INTS").StrictInts(",")
vals = cfg.Section("").Key("INT64S").StrictInt64s(",")
vals = cfg.Section("").Key("UINTS").StrictUints(",")
vals = cfg.Section("").Key("UINT64S").StrictUint64s(",")
vals = cfg.Section("").Key("TIMES").StrictTimes(",")
```

Recursive Values

For all value of keys, there is a special syntax `%(<name>)s`, where `<name>` is the key name in same section or default section, and `%(<name>)s` will be replaced by corresponding value(empty string if key not found). You can use this syntax at most 99 level of recursions.

```
NAME = ini

[author]
NAME = Unknwon
GITHUB = https://github.com/%(NAME)s

[package]
FULL_NAME = github.com/go-ini/%(NAME)s
```

```
cfg.Section("author").Key("GITHUB").String()        // https://github.com/Unknwon
cfg.Section("package").Key("FULL_NAME").String()    // github.com/go-ini/ini
```

Python Multiline values

In case, you migrate service from Python and has some legacy configurations, don’t panic!

```
cfg, err := ini.LoadSources(ini.LoadOptions{
    AllowPythonMultilineValues: true,
}, []byte(`
[long]
long_rsa_private_key = -----BEGIN RSA PRIVATE KEY-----
   foo
   bar
   foobar
   barfoo
   -----END RSA PRIVATE KEY-----
`)

/*
-----BEGIN RSA PRIVATE KEY-----
foo
bar
foobar
barfoo
-----END RSA PRIVATE KEY-----
*/
```

> 原文: *https://ini.unknwon.io/docs/howto/work_with_values*

操作注释（Comment）

Working with comments

Take care that following format will be treated as comment:

- Line begins with # or ;
- Words after # or ;
- Words after section name (i.e words after [some section name])
  If you want to save a value with `#` or `;`, please quote them with `` ` `` or `"""`.

Alternatively, you can use following LoadOptions to completely ignore inline comments:

```
1.  cfg, err := ini.LoadSources(ini.LoadOptions{
2.      IgnoreInlineComment: true,
3.  }, "app.ini")
```

> 原文: *https://ini.unknwon.io/docs/howto/work_with_comments*

高级用法

结构体与分区双向映射

Map To Struct

Want more objective way to play with INI? Cool.

```
Name = Unknwon
age = 21
Male = true
Born = 1993-01-01T20:17:05Z

[Note]
Content = Hi is a good man!
Cities = HangZhou, Boston
```

```
type Note struct {
    Content string
    Cities  []string
}

type Person struct {
    Name string
    Age  int `ini:"age"`
    Male bool
    Born time.Time
    Note
    Created time.Time `ini:"-"`
}

func main() {
    cfg, err := ini.Load("path/to/ini")
    // ...
    p := new(Person)
    err = cfg.MapTo(p)
    // ...

    // Things can be simpler.
    err = ini.MapTo(p, "path/to/ini")
    // ...

```

```
    // Just map a section? Fine.
    n := new(Note)
    err = cfg.Section("Note").MapTo(n)
    // ...
}
```

Can I have default value for field? Absolutely.

Assign it before you map to struct. It will keep the value as it is if the key is not presented or got wrong type.

```
// ...
p := &Person{
    Name: "Joe",
}
// ...
```

It’s really cool, but what’s the point if you can’t give me my file back from struct?

Reflect From Struct

Why not?

```
type Embeded struct {
    Dates  []time.Time `delim:"|" comment:"Time data"`
    Places []string    `ini:"places,omitempty"`
    None   []int       `ini:",omitempty"`
}

type Author struct {
    Name      string `ini:"NAME"`
    Male      bool
    Age       int `comment:"Author's age"`
    GPA       float64
    NeverMind string `ini:"-"`
    *Embeded `comment:"Embeded section"`
}

func main() {
    a := &Author{"Unknwon", true, 21, 2.8, "",
        &Embeded{
```

```
            []time.Time{time.Now(), time.Now()},
            []string{"HangZhou", "Boston"},
            []int{},
        }}
    cfg := ini.Empty()
    err = ini.ReflectFrom(cfg, a)
    // ...
}
```

So, what do I get?

```
NAME = Unknwon
Male = true
; Author's age
Age = 21
GPA = 2.8

; Embeded section
[Embeded]
; Time data
Dates = 2015-08-07T22:14:22+08:00|2015-08-07T22:14:22+08:00
places = HangZhou,Boston
```

Map with ShadowLoad

If you want to map a section to a struct along with ShadowLoad, then you need to indicate `allowshadow` in the struct tag.

For example, suppose you have the following configuration:

```
[IP]
value = 192.168.31.201
value = 192.168.31.211
value = 192.168.31.221
```

You should define your struct as follows:

```
type IP struct {
   Value    []string `ini:"value,omitempty,allowshadow"`
}
```

In case you don’t need the first two tag rules, then you can just have `ini:",,allowshadow"` .

Other Notes On Map/Reflect

Any embedded struct is treated as a section by default, and there is no automatic parent-child relations in map/reflect feature:

```
type Child struct {
    Age string
}

type Parent struct {
    Name string
    Child
}

type Config struct {
    City string
    Parent
}
```

Example configuration:

```
City = Boston

[Parent]
Name = Unknwon

[Child]
Age = 21
```

What if, yes, I’m paranoid, I want embedded struct to be in the same section. Well, all roads lead to Rome.

```
type Child struct {
    Age string
}

type Parent struct {
    Name string
    Child `ini:"Parent"`
```

```
}

type Config struct {
    City string
    Parent
}
```

Example configuration:

```
City = Boston

[Parent]
Name = Unknwon
Age = 21
```

See also Customize name and value mappers.

> 原文: *https://ini.unknwon.io/docs/advanced/map_and_reflect*

自定义键名和键值映射器

Name Mapper

To save your time and make your code cleaner, this library supports `NameMapper` between struct field and actual section and key name.

There are 2 built-in name mappers:

- AllCapsUnderscore: it converts to format ALL_CAPS_UNDERSCORE then match section or key.
- TitleUnderscore: it converts to format title_underscore then match section or key.
  To use them:

```
type Info struct {
    PackageName string
}

func main() {
    err = ini.MapToWithMapper(&Info{}, ini.TitleUnderscore,
[]byte("package_name=ini"))
    // ...

    cfg, err := ini.Load([]byte("PACKAGE_NAME=ini"))
    // ...
    info := new(Info)
    cfg.NameMapper = ini.AllCapsUnderscore
    err = cfg.MapTo(info)
    // ...
}
```

Same rules of name mapper apply to `ini.ReflectFromWithMapper` function.

Value Mapper

To expand values (e.g. from environment variables), you can use the `ValueMapper` to transform values:

```
type Env struct {
    Foo string `ini:"foo"`
```

```
}

func main() {
    cfg, err := ini.Load([]byte("[env]\nfoo = ${MY_VAR}\n")
    cfg.ValueMapper = os.ExpandEnv
    // ...
    env := &Env{}
    err = cfg.Section("env").MapTo(env)
}
```

This would set the value of `env.Foo` to the value of the environment variable `MY_VAR` .

> 原文: *https://ini.unknwon.io/docs/advanced/name_and_value_mapper*

常见问题

FAQs

What does BlockMode field do?

By default, library lets you read and write values so we need a locker to make sure your data is safe. But in cases that you are very sure about only reading data through the library, you can set `cfg.BlockMode = false` to speed up read operations about **50-70%** faster.

Why another INI library?

Many people are using my another INI library goconfig, so the reason for this one is I would like to make more Go style code. Also when you set `cfg.BlockMode = false`, this one is about **10-30%** faster.

To make those changes I have to confirm API broken, so it’s safer to keep it in another place and start using `gopkg.in` to version my package at this time.(PS: shorter import path)

> 原文: *https://ini.unknwon.io/docs/faqs*