# viper 简介


**coder**

keep coding




## 什么是 viper ?

它是配置解析器，负责加载并解析配置。

### 配置位置的优先级问题

Viper 可以从不同的位置读取配置，不同位置的配置具有不同的优先级，高优先级的配置会覆盖低优先级相同的配置。

1. 通过 viper.Set 函数显示设置的配置
2. 命令行参数
3. 环境变量
4. 配置文件 或 io.Reader
5. 存储默认值

### 验证配置位置的优先级

```
package main

import (
    "fmt"
    "github.com/spf13/pflag"
    "github.com/spf13/viper"
    "gopkg.in/yaml.v2"
    "strings"
)


func main () {
    var (
        hostSet string
        hostCommand string
    )
    pflag.StringVar(&hostSet, "host-set", "127.0.0.1", "")
    pflag.StringVar(&hostCommand, "host-command", "127.0.0.1", "")
    pflag.Parse()

    // 优先级最高
    viper.Set("host-set", "host-set")

    // 绑定命令行参数
    viper.BindPFlags(pflag.CommandLine)

    // 绑定环境变量
    viper.AutomaticEnv()
    // viper.Get("a-b"), 那么
    // export 环境变量时，无法使用横杠，常见分隔符是下划线
    // viper.SetEnvKeyReplacer 使得 viper.Get("host-env") 对应的环境变量为 HOST_ENV
    viper.SetEnvKeyReplacer(strings.NewReplacer(".", "_", "-", "_"))
```

```
    // 设置配置文件信息，io.Reader 方式读取配置时，也需要这些配置
    viper.AddConfigPath(".")    // // 把当前目录加入到配置文件的搜索路径中，可调用多次添加多个搜索路径
    viper.SetConfigName("config2")
    viper.SetConfigType("yaml")

    // 搜索配置文件，获取配置
    if err := viper.ReadInConfig(); err != nil {
        panic(err)
    }

    // 从 io.Reader 获取配置
    viper.ReadConfig(strings.NewReader(`
host-set:     host-set-reader
host-command: host-command-reader
host-env: host-env-reader
host-reader: host-reader-reader
`))

    // 设置默认值
    viper.SetDefault("host-set", "host-set-default")
    viper.SetDefault("host-command", "host-command-default")
    viper.SetDefault("host-env", "host-env-default")
    viper.SetDefault("host-config", "host-config-default")
    viper.SetDefault("host-reader", "host-reader-default")
    viper.SetDefault("host-default", "host-default-default")

    fmt.Println(viper.Get("host-set"))
    fmt.Println(viper.Get("host-command"))
    fmt.Println(viper.Get("host-env"))
    fmt.Println(viper.Get("host-config"))
    fmt.Println(viper.Get("host-reader"))
    fmt.Println(viper.Get("host-default"))

    // 序列化
    c := viper.AllSettings()
    bs, err := yaml.Marshal(c)
    if err != nil {
        panic(err)
    }
    fmt.Println("===== 序列化 =====")
    fmt.Println(string(bs))
}
```

执行

```
➜  viper git:(docs/read-by-ydx) X export HOST_COMMAND=host-command-command
➜  viper git:(docs/read-by-ydx) X export HOST_SET=host-set-command
➜  viper git:(docs/read-by-ydx) X export HOST_ENV=host-env-env

➜  viper git:(docs/read-by-ydx) X cat config2.yaml
host-set:     host-set-config-file
host-command: host-command-config-file
host-env: host-env-config-file
host-config: host-config-config-file

# 因为 viper.ReadConfig 在后面执行，完全替代了 viper.ReadInConfig 的解析结果。
# 即，viper.ReadInConfig 的执行不起任何作用。
➜  viper git:(docs/read-by-ydx) X go build main2.go; ./main2 --host-set=host-command-set --host-command=host-command-cc
host-set
host-command-command
```

```
host-env-env
host-config-default
host-reader-reader
host-default-default
===== 序列化 =====
host-command: host-command-command
host-config: host-config-default
host-default: host-default-default
host-env: host-env-env
host-reader: host-reader-reader
host-set: host-set
```

### viper 如何监听配置文件的变更，实时读取?

```go
package main

import (
    "fmt"
    "github.com/fsnotify/fsnotify"
    "github.com/spf13/viper"
    "path/filepath"
)

func main () {
    // 读取配置文件
    viper.AddConfigPath(".")    // // 把当前目录加入到配置文件的搜索路径中，可调用多次添加多个搜索路径
    viper.SetConfigName("config3")
    viper.SetConfigType("yaml")
    if err := viper.ReadInConfig(); err != nil {
        panic(err)
    }

    viper.WatchConfig()
    viper.OnConfigChange(func (e fsnotify.Event){
        fmt.Println("Config file changed: ", filepath.Base(e.Name))
        fmt.Println(viper.Get("host"))
    })

    select { }
}
```

执行

```
➜  viper git:(docs/read-by-ydx) X ./main3
Config file changed:  config3.yaml
host2
Config file changed:  config3.yaml
host1
```

### viper 如何支持结构化配置?

### 按照点号分割读取结构化配置某一项

```go
package main

import (
```

```
    "fmt"
    "github.com/spf13/viper"
)

func main () {
    // 读取配置文件
    viper.AddConfigPath(".")
    viper.SetConfigName("config4")
    viper.SetConfigType("json")
    if err := viper.ReadInConfig(); err != nil {
        panic(err)
    }

    fmt.Println(viper.Get("datastore.metric.host"))
    // 如果键不存在，viper.Get 返回零值，IsSet 判断是否存在该键
    fmt.Println(viper.IsSet("datastore.metric.host"))
}
```

**通过反序列化方法，解析到结构体中**

```
package main

import (
    "fmt"
    "github.com/spf13/viper"
)

type Config struct {
    Host Host
}
type Host struct {
    Address string `mapstructure:"address"`
    Port int `mapstructure:"port"`
}

func main () {
    // 读取配置文件
    viper.AddConfigPath(".")
    viper.SetConfigName("config4")
    viper.SetConfigType("json")
    if err := viper.ReadInConfig(); err != nil {
        panic(err)
    }

    var c Config
    err := viper.Unmarshal(&c)
    if err != nil {
        panic(err)
    }

    fmt.Println(c)
}
➜  viper git:(docs/read-by-ydx) ✗ cat config4.json
  {
    "datastore.metric.host": "0.0.0.0",
    "host": {
      "address": "localhost",
      "port": 5799
    },
```

```
    "datastore": {
      "metric": {
        "host": "127.0.0.1",
        "port": 3099
      },
      "warehouse": {
        "host": "198.0.0.1",
        "port": 2112
      }
    }
  }%

➜  viper git:(docs/read-by-ydx) ✗ go build main4.go; ./main4
{{localhost 5799}}
```

Viper 在后台使用 github.com/mitchellh/ma... 来解析值，其默认情况下使用 mapstructure tags。当我们需要将 Viper 读取的配置反序列到我们定义的结构体变量中时，一定要使用 **mapstructure tags。**

编辑于 2021-09-26 00:50