简介

上一篇文章介绍 cobra 的时候提到了 viper，今天我们就来介绍一下这个库。 viper 是一个配置解决方案，拥有丰富的特性：

- 支持 JSON/TOML/YAML/HCL/envfile/Java properties 等多种格式的配置文件；
- 可以设置监听配置文件的修改，修改时自动加载新的配置；
- 从环境变量、命令行选项和 `io.Reader` 中读取配置；
- 从远程配置系统中读取和监听修改，如 etcd/Consul；
- 代码逻辑中显示设置键值。

快速使用

安装：

```
$ go get github.com/spf13/viper
```

使用：

```go
package main

import (
  "fmt"
  "log"

  "github.com/spf13/viper"
)

func main() {
  viper.SetConfigName("config")
  viper.SetConfigType("toml")
  viper.AddConfigPath(".")
  viper.SetDefault("redis.port", 6381)
  err := viper.ReadInConfig()
  if err != nil {
    log.Fatal("read config failed: %v", err)
  }

  fmt.Println(viper.Get("app_name"))
  fmt.Println(viper.Get("log_level"))

  fmt.Println("mysql ip: ", viper.Get("mysql.ip"))
  fmt.Println("mysql port: ", viper.Get("mysql.port"))
  fmt.Println("mysql user: ", viper.Get("mysql.user"))
  fmt.Println("mysql password: ", viper.Get("mysql.password"))
  fmt.Println("mysql database: ", viper.Get("mysql.database"))

  fmt.Println("redis ip: ", viper.Get("redis.ip"))
  fmt.Println("redis port: ", viper.Get("redis.port"))
}
```

我们使用之前Go 每日一库之 go-ini一文中使用的配置，不过改为 toml 格式。 toml 的语法很简单，快速入门请看learn X in Y minutes。

```toml
app_name = "awesome web"

# possible values: DEBUG, INFO, WARNING, ERROR, FATAL
log_level = "DEBUG"

[mysql]
ip = "127.0.0.1"
port = 3306
user = "dj"
password = 123456
database = "awesome"

[redis]
ip = "127.0.0.1"
port = 7381
```

viper 的使用非常简单，它需要很少的设置。设置文件名（`SetConfigName`）、配置类型（`SetConfigType`）和搜索路径（`AddConfigPath`），然后调用 `ReadInConfig`。 viper会自动根据类型来读取配置。使用时调用 `viper.Get` 方法获取键值。

编译、运行程序：

```
awesome web
DEBUG
mysql ip:  127.0.0.1
mysql port:  3306
mysql user:  dj
mysql password:  123456
mysql database:  awesome
redis ip:  127.0.0.1
redis port:  7381
```

有几点需要注意：

- 设置文件名时不要带后缀；
- 搜索路径可以设置多个，viper 会根据设置顺序依次查找；
- viper 获取值时使用 `section.key` 的形式，即传入嵌套的键名；
- 默认值可以调用 `viper.SetDefault` 设置。

读取键

viper 提供了多种形式的读取方法。在上面的例子中，我们看到了 `Get` 方法的用法。 `Get` 方法返回一个 `interface{}` 的值，使用有所不便。

`GetType` 系列方法可以返回指定类型的值。 其中，Type 可以为 `Bool/Float64/Int/String/Time/Duration/IntSlice/StringSlice`。 但是请注意，**如果指定的键不存在或类型不正确， `GetType` 方法返回对应类型的零值。**

如果要判断某个键是否存在，使用 `IsSet` 方法。 另外， `GetStringMap` 和 `GetStringMapString` 直接以 map 返回某个键下面所有的键值对，前者返回 `map[string]interface{}`，后者返回 `map[string]string`。 `AllSettings` 以 `map[string]interface{}` 返回所有设置。

```go
// 省略包名和 import 部分

func main() {
  viper.SetConfigName("config")
  viper.SetConfigType("toml")
  viper.AddConfigPath(".")
  err := viper.ReadInConfig()
  if err != nil {
    log.Fatal("read config failed: %v", err)
  }

  fmt.Println("protocols: ", viper.GetStringSlice("server.protocols"))
  fmt.Println("ports: ", viper.GetIntSlice("server.ports"))
  fmt.Println("timeout: ", viper.GetDuration("server.timeout"))

  fmt.Println("mysql ip: ", viper.GetString("mysql.ip"))
  fmt.Println("mysql port: ", viper.GetInt("mysql.port"))

  if viper.IsSet("redis.port") {
    fmt.Println("redis.port is set")
  } else {
    fmt.Println("redis.port is not set")
  }

  fmt.Println("mysql settings: ", viper.GetStringMap("mysql"))
  fmt.Println("redis settings: ", viper.GetStringMap("redis"))
  fmt.Println("all settings: ", viper.AllSettings())
}
```

我们在配置文件 config.toml 中添加 `protocols` 和 `ports` 配置：

```toml
[server]
protocols = ["http", "https", "port"]
ports = [10000, 10001, 10002]
timeout = 3s
```

编译、运行程序，输出：

```
protocols:  [http https port]
ports:  [10000 10001 10002]
timeout:  3s
mysql ip:  127.0.0.1
mysql port:  3306
redis.port is set
mysql settings:  map[database:awesome ip:127.0.0.1 password:123456 port:3306 user:dj]
redis settings:  map[ip:127.0.0.1 port:7381]
all settings:  map[app_name:awesome web log_level:DEBUG mysql:map[database:awesome ip:
```

如果将配置中的 `redis.port` 注释掉，将输出 `redis.port is not set`。

上面的示例中还演示了如何使用 `time.Duration` 类型，只要是 `time.ParseDuration` 接受的格式都可以，例如 `3s`、 `2min`、 `1min30s` 等。

设置键值

viper 支持在多个地方设置，使用下面的顺序依次读取：

- 调用 `Set` 显示设置的；
- 命令行选项；
- 环境变量；
- 配置文件；
- 默认值。

`viper.Set`

如果某个键通过 `viper.Set` 设置了值，那么这个值的优先级最高。

```go
viper.Set("redis.port", 5381)
```

如果将上面这行代码放到程序中，运行程序，输出的 `redis.port` 将是 5381。

命令行选项

如果一个键没有通过 `viper.Set` 显示设置值，那么获取时将尝试从命令行选项中读取。 如果有，优先使用。viper 使用 pflag 库来解析选项。 我们首先在 `init` 方法中定义选项，并且调用 `viper.BindPFlags` 绑定选项到配置中：

```go
func init() {
  pflag.Int("redis.port", 8381, "Redis port to connect")

  // 绑定命令行
  viper.BindPFlags(pflag.CommandLine)
}
```

然后，在 `main` 方法开头处调用 `pflag.Parse` 解析选项。

编译、运行程序：

```
$ ./main.exe --redis.port 9381
awesome web
DEBUG
mysql ip:  127.0.0.1
mysql port:  3306
mysql user:  dj
mysql password:  123456
mysql database:  awesome
redis ip:  127.0.0.1
redis port:  9381
```

如何不传入选项：

```
$ ./main.exe
awesome web
DEBUG
mysql ip:  127.0.0.1
mysql port:  3306
mysql user:  dj
mysql password:  123456
mysql database:  awesome
redis ip:  127.0.0.1
redis port:  7381
```

注意，这里并不会使用选项 `redis.port` 的默认值。

但是，如果通过下面的方法都无法获得键值，那么返回选项默认值（如果有）。试试注释掉配置文件中 `redis.port` 看看效果。

环境变量

如果前面都没有获取到键值，将尝试从环境变量中读取。我们既可以一个个绑定，也可以自动全部绑定。

在 `init` 方法中调用 `AutomaticEnv` 方法绑定全部环境变量：

```go
func init() {
  // 绑定环境变量
  viper.AutomaticEnv()
}
```

为了验证是否绑定成功，我们在 `main` 方法中将环境变量 GOPATH 打印出来：

```go
func main() {
  // 省略部分代码

  fmt.Println("GOPATH: ", viper.Get("GOPATH"))
}
```

通过 系统 -> 高级设置 -> 新建 创建一个名为 `redis.port` 的环境变量，值为 10381。 运行程序，输出的 `redis.port` 值为 10381，并且输出中有 GOPATH 信息。

也可以单独绑定环境变量：

```go
func init() {
  // 绑定环境变量
  viper.BindEnv("redis.port")
  viper.BindEnv("go.path", "GOPATH")
}

func main() {
  // 省略部分代码
  fmt.Println("go path: ", viper.Get("go.path"))
}
```

调用 `BindEnv` 方法，如果只传入一个参数，则这个参数既表示键名，又表示环境变量名。 如果传入两个参数，则第一个参数表示键名，第二个参数表示环境变量名。

还可以通过 `viper.SetEnvPrefix` 方法设置环境变量前缀，这样一来，通过 `AutomaticEnv` 和一个参数的 `BindEnv` 绑定的环境变量， 在使用 `Get` 的时候，viper 会自动加上这个前缀再从环境变量中查找。

如果对应的环境变量不存在，viper 会自动将键名全部转为大写再查找一次。所以，使用键名 `gopath` 也能读取环境变量 `GOPATH` 的值。

配置文件

如果经过前面的途径都没能找到该键，viper 接下来会尝试从配置文件中查找。 为了避免环境变量的影响，需要删除 `redis.port` 这个环境变量。

看快速使用中的示例。

默认值

在上面的快速使用一节，我们已经看到了如何设置默认值，这里就不赘述了。

读取配置

从 `io.Reader` 中读取

viper 支持从 `io.Reader` 中读取配置。这种形式很灵活，来源可以是文件，也可以是程序中生成的字符串，甚至可以从网络连接中读取的字节流。

```go
package main

import (
  "bytes"
  "fmt"
  "log"

  "github.com/spf13/viper"
)

func main() {
  viper.SetConfigType("toml")
  tomlConfig := []byte(`
app_name = "awesome web"

# possible values: DEBUG, INFO, WARNING, ERROR, FATAL
log_level = "DEBUG"

[mysql]
ip = "127.0.0.1"
port = 3306
user = "dj"
password = 123456
database = "awesome"

[redis]
ip = "127.0.0.1"
port = 7381
`)
  err := viper.ReadConfig(bytes.NewBuffer(tomlConfig))
  if err != nil {
    log.Fatal("read config failed: %v", err)
  }

  fmt.Println("redis port: ", viper.GetInt("redis.port"))
}
```

`Unmarshal`

viper 支持将配置 `Unmarshal` 到一个结构体中，为结构体中的对应字段赋值。

```go
package main

import (
  "fmt"
  "log"

  "github.com/spf13/viper"
)

type Config struct {
  AppName  string
  LogLevel string

  MySQL    MySQLConfig
  Redis    RedisConfig
}

type MySQLConfig struct {
  IP       string
  Port     int
  User     string
  Password string
  Database string
}

type RedisConfig struct {
  IP   string
  Port int
}

func main() {
  viper.SetConfigName("config")
  viper.SetConfigType("toml")
  viper.AddConfigPath(".")
  err := viper.ReadInConfig()
  if err != nil {
    log.Fatal("read config failed: %v", err)
  }

  var c Config
  viper.Unmarshal(&c)

  fmt.Println(c.MySQL)
}
```

编译，运行程序，输出：

```
{127.0.0.1 3306 dj 123456 awesome}
```

保存配置

有时候，我们想要将程序中生成的配置，或者所做的修改保存下来。viper 提供了接口！

- `WriteConfig`：将当前的 viper 配置写到预定义路径，如果没有预定义路径，返回错误。将会覆盖当前配置；
- `SafeWriteConfig`：与上面功能一样，但是如果配置文件存在，则不覆盖；
- `WriteConfigAs`：保存配置到指定路径，如果文件存在，则覆盖；
- `SafeWriteConfig`：与上面功能一样，但是入股配置文件存在，则不覆盖。

下面我们通过程序生成一个 `config.toml` 配置：

```go
package main

import (
  "log"

  "github.com/spf13/viper"
)

func main() {
  viper.SetConfigName("config")
  viper.SetConfigType("toml")
  viper.AddConfigPath(".")

  viper.Set("app_name", "awesome web")
  viper.Set("log_level", "DEBUG")
  viper.Set("mysql.ip", "127.0.0.1")
  viper.Set("mysql.port", 3306)
  viper.Set("mysql.user", "root")
  viper.Set("mysql.password", "123456")
  viper.Set("mysql.database", "awesome")

  viper.Set("redis.ip", "127.0.0.1")
  viper.Set("redis.port", 6381)

  err := viper.SafeWriteConfig()
  if err != nil {
    log.Fatal("write config failed: ", err)
  }
}
```

编译、运行程序，生成的文件如下：

```toml
app_name = "awesome web"
log_level = "DEBUG"

[mysql]
  database = "awesome"
  ip = "127.0.0.1"
  password = "123456"
  port = 3306
  user = "root"

[redis]
  ip = "127.0.0.1"
  port = 6381
```

监听文件修改

viper 可以监听文件修改，热加载配置。因此不需要重启服务器，就能让配置生效。

```go
package main

import (
  "fmt"
  "log"
  "time"

  "github.com/spf13/viper"
)

func main() {
  viper.SetConfigName("config")
  viper.SetConfigType("toml")
  viper.AddConfigPath(".")
  err := viper.ReadInConfig()
  if err != nil {
    log.Fatal("read config failed: %v", err)
  }

  viper.WatchConfig()

  fmt.Println("redis port before sleep: ", viper.Get("redis.port"))
  time.Sleep(time.Second * 10)
  fmt.Println("redis port after sleep: ", viper.Get("redis.port"))
}
```

只需要调用 `viper.WatchConfig`，viper 会自动监听配置修改。如果有修改，重新加载的配置。

上面程序中，我们先打印 `redis.port` 的值，然后 `Sleep` 10s。在这期间修改配置中 `redis.port` 的值， `Sleep` 结束后再次打印。 发现打印出修改后的值：

```
redis port before sleep:  7381
redis port after sleep:  73810
```

另外，还可以为配置修改增加一个回调：

```go
viper.OnConfigChange(func(e fsnotify.Event) {
  fmt.Printf("Config file:%s Op:%s\n", e.Name, e.Op)
})
```

这样文件修改时会执行这个回调。

viper 使用fsnotify这个库来实现监听文件修改的功能。

完整示例代码见 GitHub。

参考

1. viper GitHub 仓库