链滴

解析 cobra

作者：someone27889
原文链接：https://ld246.com/article/1587518015660
来源网站：链滴

```go
// Copyright © 2015 Steve Francia <spf@spf13.com>.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
// http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import (
    "os"

    "github.com/spf13/cobra/cobra/cmd"
)

func main() {
    if err := cmd.Execute(); err != nil {
        os.Exit(1)
    }
}
```

main.go 调用了 cmd 的 Excute 函数,如果没有错误信息则正常退出

```go
var (
    // Used for flags.
    cfgFile     string
    userLicense string

    rootCmd = &cobra.Command{
        Use:   "cobra",
        Short: "A generator for Cobra based Applications",
        Long: `Cobra is a CLI library for Go that empowers applications.
This application is a tool to generate the needed files
to quickly create a Cobra application.`,
    }
)
```

首先定义了一些变量,cfgFile,userLicense,rootCmd

其中 rootCmd 是指针类型的 Command 结构体,传入了 Use,Short,Long,看一下 cobra 包下的 Command 结构体类型

```go
type Command struct {
    // Use is the one-line usage message.
    Use string

    // Aliases is an array of aliases that can be used instead of the first word in Use.
```

```
    Aliases []string

    // SuggestFor is an array of command names for which this command will be suggested -
    // similar to aliases but only suggests.
    SuggestFor []string

    // Short is the short description shown in the 'help' output.
    Short string

    // Long is the long message shown in the 'help <this-command>' output.
    Long string

    // Example is examples of how to use the command.
    Example string

    // ValidArgs is list of all valid non-flag arguments that are accepted in bash completions
    ValidArgs []string
    // ValidArgsFunction is an optional function that provides valid non-flag arguments for bas
 completion.
    // It is a dynamic version of using ValidArgs.
    // Only one of ValidArgs and ValidArgsFunction can be used for a command.
    ValidArgsFunction func(cmd *Command, args []string, toComplete string) ([]string, ShellCo
pDirective)

    // Expected arguments
    Args PositionalArgs

    // ArgAliases is List of aliases for ValidArgs.
    // These are not suggested to the user in the bash completion,
    // but accepted if entered manually.
    ArgAliases []string

    // BashCompletionFunction is custom functions used by the bash autocompletion generato
.
    BashCompletionFunction string

    // Deprecated defines, if this command is deprecated and should print this string when use
.
    Deprecated string

    // Hidden defines, if this command is hidden and should NOT show up in the list of availab
e commands.
    Hidden bool

    // Annotations are key/value pairs that can be used by applications to identify or
    // group commands.
    Annotations map[string]string

    // Version defines the version for this command. If this value is non-empty and the comma
d does not
    // define a "version" flag, a "version" boolean flag will be added to the command and, if sp
cified,
    // will print content of the "Version" variable. A shorthand "v" flag will also be added if the
    // command does not define one.
```

```
    Version string

    // The *Run functions are executed in the following order:
    //   * PersistentPreRun()
    //   * PreRun()
    //   * Run()
    //   * PostRun()
    //   * PersistentPostRun()
    // All functions get the same args, the arguments after the command name.
    //
    // PersistentPreRun: children of this command will inherit and execute.
    PersistentPreRun func(cmd *Command, args []string)
    // PersistentPreRunE: PersistentPreRun but returns an error.
    PersistentPreRunE func(cmd *Command, args []string) error
    // PreRun: children of this command will not inherit.
    PreRun func(cmd *Command, args []string)
    // PreRunE: PreRun but returns an error.
    PreRunE func(cmd *Command, args []string) error
    // Run: Typically the actual work function. Most commands will only implement this.
    Run func(cmd *Command, args []string)
    // RunE: Run but returns an error.
    RunE func(cmd *Command, args []string) error
    // PostRun: run after the Run command.
    PostRun func(cmd *Command, args []string)
    // PostRunE: PostRun but returns an error.
    PostRunE func(cmd *Command, args []string) error
    // PersistentPostRun: children of this command will inherit and execute after PostRun.
    PersistentPostRun func(cmd *Command, args []string)
    // PersistentPostRunE: PersistentPostRun but returns an error.
    PersistentPostRunE func(cmd *Command, args []string) error

    // SilenceErrors is an option to quiet errors down stream.
    SilenceErrors bool

    // SilenceUsage is an option to silence usage when an error occurs.
    SilenceUsage bool

    // DisableFlagParsing disables the flag parsing.
    // If this is true all flags will be passed to the command as arguments.
    DisableFlagParsing bool

    // DisableAutoGenTag defines, if gen tag ("Auto generated by spf13/cobra...")
    // will be printed by generating docs for this command.
    DisableAutoGenTag bool

    // DisableFlagsInUseLine will disable the addition of [flags] to the usage
    // line of a command when printing help or generating docs
    DisableFlagsInUseLine bool

    // DisableSuggestions disables the suggestions based on Levenshtein distance
    // that go along with 'unknown command' messages.
    DisableSuggestions bool
    // SuggestionsMinimumDistance defines minimum levenshtein distance to display suggest
ons.
```

```
// Must be > 0.
SuggestionsMinimumDistance int

// TraverseChildren parses flags on all parents before executing child command.
TraverseChildren bool

// FParseErrWhitelist flag parse errors to be ignored
FParseErrWhitelist FParseErrWhitelist

ctx context.Context

// commands is the list of commands supported by this program.
commands []*Command
// parent is a parent command for this command.
parent *Command
// Max lengths of commands' string lengths for use in padding.
commandsMaxUseLen         int
commandsMaxCommandPathLen int
commandsMaxNameLen        int
// commandsAreSorted defines, if command slice are sorted or not.
commandsAreSorted bool
// commandCalledAs is the name or alias value used to call this command.
commandCalledAs struct {
    name   string
    called bool
}

// args is actual args parsed from flags.
args []string
// flagErrorBuf contains all error messages from pflag.
flagErrorBuf *bytes.Buffer
// flags is full set of flags.
flags *flag.FlagSet
// pflags contains persistent flags.
pflags *flag.FlagSet
// lflags contains local flags.
lflags *flag.FlagSet
// iflags contains inherited flags.
iflags *flag.FlagSet
// parentsPflags is all persistent flags of cmd's parents.
parentsPflags *flag.FlagSet
// globNormFunc is the global normalization function
// that we can use on every pflag set and children commands
globNormFunc func(f *flag.FlagSet, name string) flag.NormalizedName

// usageFunc is usage func defined by user.
usageFunc func(*Command) error
// usageTemplate is usage template defined by user.
usageTemplate string
// flagErrorFunc is func defined by user and it's called when the parsing of
// flags returns an error.
flagErrorFunc func(*Command, error) error
// helpTemplate is help template defined by user.
helpTemplate string
```

```
    // helpFunc is help func defined by user.
    helpFunc func(*Command, []string)
    // helpCommand is command with usage 'help'. If it's not defined by user,
    // cobra uses default help command.
    helpCommand *Command
    // versionTemplate is the version template defined by user.
    versionTemplate string

    // inReader is a reader defined by the user that replaces stdin
    inReader io.Reader
    // outWriter is a writer defined by the user that replaces stdout
    outWriter io.Writer
    // errWriter is a writer defined by the user that replaces stderr
    errWriter io.Writer
}
```

Command 结构体中定义了好多属性，挨个看看都是干嘛的

variable	type	description
Use	String	一行如何使用 Command 的信息
Aliases	[]string	Command 的名数组
SuggestFor	[]string	建议使用命令的命令名称数组,建议
Short	[]string	帮助输出中显示简短描述。
Long	[]string	帮助输出中显示长 miao shu
Example	string	如何使用 Command 的信息
ValidArgs	[]string	bash 中接受所有有效,非标志参数的列表
ValidArgsFunction	func(*Command,args[]string,toComplete string)([]string,ShellCompDirective)	和 ValidArgs 相同，但提供函数二者只能选一种
Args	PositionalArgs	预期参数
ArgAliases	[]string	ValidArgs 别名列表
BashCompletionFunction	string	bash 自动完成生成器使用的自定义函数
Deprecated	string	弃用信息
Hidden	bool	是否隐藏
Annotations	map[string]string	供应用程序用来识别或分组
Version	string	版本号

PresistentPreRun func(cmd*Command,args []string)
命令的子级将继承并执行。

PresistentPreRunE func(cmd *Command,args []string)error
命令的子级将继承并执行，返回 error

PreRun func(cmd *Command,args []string)
命令的子级将不会继承

PreRunE func(cmd *Command,args []string)error
上但返回 error

PostRun func(cmd *Command,args []string)
Run 后执行 Command

PostRunE

Run func(cmd *Command,args []string)
un

RunE

PresistenPostRun func(cmd *Command,args []string)

PresistenPostRunE

SilenceErrors bool

SilenceUsage bool

DisableFlagParsing bool 禁用 fl
g 解析

DsiableAutoGenTag bool 是否
过生成此命令的文档来打印 gen 标签

DsiableFlagsInUseLine bool
打印帮助或生成文档时，将禁止在命令的行中添加[flags]

DisableSuggestions bool

SuggestionsMiniumDistance int

TravereChildren bool 在执行子
令之前解析所有父项上的标志。

FParseErrWhitelist FParseErrWhitelist
志解析要忽略的错误

ctx context.Context

commands []*Command 程序
持的命令列表

parent *Command 此命令的父
令

commandsMaxUseLen int 用
填充的命令字符串最大长度

commandsMaxCommandPathLen int

commandsMaxNameLen int

commandAreSorted bool 是
对命令片排序

commandCalledAs struct{name sstring,called bool}

用此命令的名称或别名值

args	[]strings	从标志解析的实际rgs
flagErrorBuf	*bytes.Buffer	包来自 pflag 的所有错误消息。
flags	*flag.FlagSet	标志
pflags	*flag.FlagSet	persistent标志
lflags	*flag.FlagSet	local 标志
iflags	*flag.FlagSet	inherited志
parentsPflags	*flag.FlagSet	cm父母的所有永久标志
globNormFunc	func(f *flag.FlagSet,name string)flag.NormalizdName	全局函数
useageFunc	func(*Command)error	户定义的用法功能
flagErrorFunc	func(*Command,error)error	
helpTemplate	string	用户定义帮助模板
helpFunc	func(*Command,[]string)	户定义的帮助功能
helpCommand	*Command	用为“help”的命令
versionTemplate	string	用户定的版本模板
inReader	io.Reader	stdin
outWriter	io.Writer	stdout
errWriter	io.Writer	stderr

然后是 init 函数,使用 root 文件 cmd.Excute 时自动调用

```
cobra.OnInitialize(initConfig)
```

这条代码调用到

```
func SetConfigFile(in string) { v.SetConfigFile(in) }
func (v *Viper) SetConfigFile(in string) {
    if in != "" {
        v.configFile = in
    }
}
```

SetConfigFile 调用结构体 Viper 的 SetConfigFIle 函数

viper 包的 init 函数

```go
func New() *Viper {
    v := new(Viper)
    v.keyDelim = "."
    v.configName = "config"
    v.configPermissions = os.FileMode(0644)
    v.fs = afero.NewOsFs()
    v.config = make(map[string]interface{})
    v.override = make(map[string]interface{})
    v.defaults = make(map[string]interface{})
    v.kvstore = make(map[string]interface{})
    v.pflags = make(map[string]FlagValue)
    v.env = make(map[string]string)
    v.aliases = make(map[string]string)
    v.typeByDefValue = false

    return v
}
```

viper 是个什么东西呢，康康

```go
type Viper struct {
    // 分隔键列表的定界符
    // 用于一次性访问嵌套值
    keyDelim string

    // 查找配置文件的路径
    configPaths []string

    // 读取配置文件地址的类型
    fs afero.Fs

    // 一组远程提供程序以搜索配置
    remoteProviders []*defaultRemoteProvider

    // 配置文件名称
    configName        string
    configFile        string
    configType        string
    // 文件权限
    configPermissions os.FileMode
    // 环境前缀
    envPrefix         string
    automaticEnvApplied bool
    envKeyReplacer      *strings.Replacer
    allowEmptyEnv       bool

    config         map[string]interface{}
    override       map[string]interface{}
    defaults       map[string]interface{}
    kvstore        map[string]interface{}
    pflags         map[string]FlagValue
    env            map[string]string
    aliases        map[string]string
    typeByDefValue bool
```

```
    //将读取的属性存储在对象上，以便我们可以按顺序写回并带有注释。
    //仅当读取的配置是属性文件时才使用。
    properties *properties.Properties

    onConfigChange func(fsnotify.Event)
}

func SetConfigFile(in string) { v.SetConfigFile(in) }
func (v *Viper) SetConfigFile(in string) {
    if in != "" {
        // 将viper结构体重的 configFile配置成
        v.configFile = in
    }
}

func initConfig() {
    if cfgFile != "" {
        // 如果配置了配置文件地址则 viper结构体设置此文件
        viper.SetConfigFile(cfgFile)
    } else {
        // 否则从home开始找配置文件
        home, err := homedir.Dir()
        if err != nil {
            er(err)
        }
        // viper配置寻找配置文件的目录
        viper.AddConfigPath(home)
        // 寻找以.cobra结尾的文件
        viper.SetConfigName(".cobra")
    }
    // 自动判断系统环境并应用
    // 这个函数将viper.automaticEnvApplied gai we
    viper.AutomaticEnv()
    // 如果读取配置文件出错
    if err := viper.ReadInConfig(); err == nil {
        fmt.Println("Using config file:", viper.ConfigFileUsed())
    }
}
```

然后看下一句

```
rootCmd.PersistentFlags().StringVar(&cfgFile, "config", "", "config file (default is $HOME/.cobr
.yaml)")

// 返回当前命令中设置的持久性FlagSet
func (c *Command) PersistentFlags() *flag.FlagSet {
    // 如果command的pflgs-FlagSet为空
    if c.pflags == nil {
        // 那么新建一个
        c.pflags = flag.NewFlagSet(c.Name(), flag.ContinueOnError)
        if c.flagErrorBuf == nil {
            c.flagErrorBuf = new(bytes.Buffer)
        }
```

```
        c.pflags.SetOutput(c.flagErrorBuf)
    }
    // return 回这个FlagSet,指针类型
    return c.pflags
}
```

什么是flag呢,比如xxx create --name ferried,在这段command中,create为command,--name为create的flag

看一眼FlagSet 结构体

```
type FlagSet struct {
    Usage func()
    SortFlags bool
    ParseErrorsWhitelist ParseErrorsWhitelist
    name              string
    parsed            bool
    actual            map[NormalizedName]*Flag
    orderedActual     []*Flag
    sortedActual      []*Flag
    formal            map[NormalizedName]*Flag
    orderedFormal     []*Flag
    sortedFormal      []*Flag
    shorthands        map[byte]*Flag
    args              []string // arguments after flags
    argsLenAtDash     int
    errorHandling     ErrorHandling
    output            io.Writer
    interspersed      bool      args
    normalizeNameFunc func(f *FlagSet, name string) NormalizedName
    addedGoFlagSets []*goflag.FlagSet
}
```

StringVar

```
func (f *FlagSet) StringVar(p *string, name string, value string, usage string) {
    f.VarP(newStringValue(value, p), name, "", usage)
}
```

最后走到,新建一个Flag,然后加入到FlagSet中

```
func (f *FlagSet) VarPF(value Value, name, shorthand, usage string) *Flag {
    // Remember the default value as a string; it won't change.
    flag := &Flag{
        Name:      name,
        Shorthand: shorthand,
        Usage:     usage,
        Value:     value,
        DefValue:  value.String(),
    }
    f.AddFlag(flag)
    return flag
}
```

Flag结构体,一个FlagSet对应多个Flag

```
type Flag struct {
    Name              string            // name as it appears on command line
    Shorthand         string            // one-letter abbreviated flag
    Usage             string            // help message
    Value             Value             // value as set
    DefValue          string            // default value (as text); for usage message
    Changed           bool              // If the user set the value (or if left to default)
    NoOptDefVal       string            // default value (as text); if the flag is on the command lin
 without any options
    Deprecated        string            // If this flag is deprecated, this string is the new or now th
ng to use
    Hidden            bool              // used by cobra.Command to allow flags to be hidden from
help/usage text
    ShorthandDeprecated string          // If the shorthand of this flag is deprecated, this strin
 is the new or now thing to use
    Annotations       map[string][]string // used by cobra.Command bash autocomple code
}
```

至此一个flag就加入到command中了.

再往下看

```
// 这里除了StringP 其余都与 上面的函数相同
rootCmd.PersistentFlags().StringP("author", "a", "YOUR NAME", "author name for copyright at
ribution")
```

不同点在于 shorthand属性

举例

command create --name ferried

command create -n ferried

这里 --name 和 -n 分别对应StringVar,StringP

再来重新对比一下三个String函数

```
StringP
StringVar
StringVarP

func (f *FlagSet) StringP(name, shorthand string, value string, usage string) *string {
    p := new(string)
    f.StringVarP(p, name, shorthand, value, usage)
    return p
}

func (f *FlagSet) StringVar(p *string, name string, value string, usage string) {
    f.VarP(newStringValue(value, p), name, "", usage)
}

func (f *FlagSet) StringVarP(p *string, name, shorthand string, value string, usage string) {
    f.VarP(newStringValue(value, p), name, shorthand, usage)
}
```

区别就在于shorthand和*string

再往下看

```
rootCmd.PersistentFlags().Bool

func (f *FlagSet) Bool(name string, value bool, usage string) *bool {
    return f.BoolP(name, "", value, usage)
}


func (f *FlagSet) BoolP(name, shorthand string, value bool, usage string) *bool {
    p := new(bool)
    f.BoolVarP(p, name, shorthand, value, usage)
    return p
}

func (f *FlagSet) BoolVarP(p *bool, name, shorthand string, value bool, usage string) {
    flag := f.VarPF(newBoolValue(value, p), name, shorthand, usage)
    flag.NoOptDefVal = "true"
}


func (f *FlagSet) VarPF(value Value, name, shorthand, usage string) *Flag {
    // Remember the default value as a string; it won't change.
    flag := &Flag{
        Name:      name,
        Shorthand: shorthand,
        Usage:     usage,
        Value:     value,
        DefValue:  value.String(),
    }
    f.AddFlag(flag)
    return flag
}
```

也是处理一下值的转换然后shorhand不同，加一个Flag结构体到Command的pflag中

再往下看

```
// 绑定 author ,第二个参数是 查找出的flag
viper.BindPFlag("author", rootCmd.PersistentFlags().Lookup("author"))

// lookup调用了FlagSet.lookup函数,传入了normalize配置的函数来 format一下flag的name
func (f *FlagSet) Lookup(name string) *Flag {
    return f.lookup(f.normalizeFlagName(name))
}
// 最后return 出去这个Flag
func (f *FlagSet) lookup(name NormalizedName) *Flag {
    return f.formal[name]
}
// 然后调用 Viper结构体中的BindPFlag
func BindPFlag(key string, flag *pflag.Flag) error { return v.BindPFlag(key, flag) }
func (v *Viper) BindPFlag(key string, flag *pflag.Flag) error {
    return v.BindFlagValue(key, pflagValue{flag})
}
// return error或者nil
```

```go
func BindFlagValue(key string, flag FlagValue) error {
 return v.BindFlagValue(key, flag)
}

func (v *Viper) BindFlagValue(key string, flag FlagValue) error {
    // 如果 flag为空
    if flag == nil {
        // 异常信息
        return fmt.Errorf("flag for %q is nil", key)
    }
    // 如果可以找到,那么viper的pflugs这个FlagSet[key]设置成flag结构体，返回nil
    v.pflags[strings.ToLower(key)] = flag
    return nil
}
```

再往下看

```go
    // 绑定好了就可以设置默认值了
    viper.SetDefault("author", "NAME HERE <EMAIL ADDRESS>")
    viper.SetDefault("license", "apache")
```

看一下命令执行结果

```
ferried@ferrieds-mac cobra % go run cobra/main.go
Cobra is a CLI library for Go that empowers applications.
This application is a tool to generate the needed files
to quickly create a Cobra application.

Usage:
  cobra [command]

Available Commands:
  add         Add a command to a Cobra Application
  help        Help about any command
  init        Initialize a Cobra Application

Flags:
  -a, --author string    author name for copyright attribution (default "YOUR NAME")
      --config string    config file (default is $HOME/.cobra.yaml)
  -h, --help             help for cobra
  -l, --license string   name of license for the project
      --viper            use Viper for configuration (default true)

Use "cobra [command] --help" for more information about a command.
ferried@ferrieds-mac cobra %
```

```
ferried@ferrieds-mac cobra % go run cobra/main.go add --help
Add (cobra add) will create a new command, with a license and
the appropriate structure for a Cobra-based CLI application,
and register it to its parent (default rootCmd).

If you want your command to be public, pass in the command name
with an initial uppercase letter.

Example: cobra add server -> resulting in a new cmd/server.go

Usage:
  cobra add [command name] [flags]

Aliases:
  add, command

Flags:
  -h, --help            help for add
  -p, --parent string   variable name of parent command for this command (default "rootCmd")

Global Flags:
  -a, --author string    author name for copyright attribution (default "YOUR NAME")
      --config string    config file (default is $HOME/.cobra.yaml)
  -l, --license string   name of license for the project
      --viper            use Viper for configuration (default true)
```

```
ferried@ferrieds-mac cobra % go run cobra/main.go init --help
Initialize (cobra init) will create a new application, with a license
and the appropriate structure for a Cobra-based CLI application.

  * If a name is provided, it will be created in the current directory;
  * If no name is provided, the current directory will be assumed;
  * If a relative path is provided, it will be created inside $GOPATH
    (e.g. github.com/spf13/hugo);
  * If an absolute path is provided, it will be created;
  * If the directory already exists but is empty, it will be used.

Init will not use an existing directory with contents.

Usage:
  cobra init [name] [flags]

Aliases:
  init, initialize, initialise, create

Flags:
  -h, --help              help for init
      --pkg-name string   fully qualified pkg name

Global Flags:
  -a, --author string    author name for copyright attribution (default "YOUR NAME")
      --config string    config file (default is $HOME/.cobra.yaml)
  -l, --license string   name of license for the project
      --viper            use Viper for configuration (default true)
```

然后看下面两条cobra下的子command

```
rootCmd.AddCommand(addCmd)
rootCmd.AddCommand(initCmd)
```