codegangsta-cli 使用手册

0.5

疯狂的笑笑

概览

cli 是 Go 语言中一个简单高效并且有趣的，帮助开发命令行程序的小框架包。它的目标是让开发者在一种更加友好的开发体验中轻松愉快地快速编写与发布命令行程序。

通常来说，命令行程序一般都很小，但这并不应当是你的程序不能文档化的理由，而像生成帮助文档，解析命令行的参数选项等这些事情，在我们编写命令行程序的时候却又影响我们的开发生产效率。

cli 正是为了解决这些问题而来，它使命令行程序的开发更加有趣，文档组织更加规范，条理清楚。

安装

Make sure you have a working Go environment (go 1.1+ is *required*). See the install instructions.

To install `cli.go`, simply run:

```
$ go get github.com/codegangsta/cli
```

Make sure your `PATH` includes to the `$GOPATH/bin` directory so your commands can be easily used:

```
export PATH=$PATH:$GOPATH/bin
```

起步

One of the philosophies behind `cli.go` is that an API should be playful and full of discovery. So a `cli.go` app can be as little as one line of code in `main()`.

```
package main
```

```
import (
  "os"
  "github.com/codegangsta/cli"
)

func main() {
  cli.NewApp().Run(os.Args)
}
```

This app will run and show help text, but is not very useful. Let's give an action to execute and some help documentation:

```
package main

import (
  "os"
  "github.com/codegangsta/cli"
)

func main() {
  app := cli.NewApp()
  app.Name = "boom"
  app.Usage = "make an explosive entrance"
  app.Action = func(c *cli.Context) {
    println("boom! I say!")
  }

  app.Run(os.Args)
}
```

Running this already gives you a ton of functionality, plus support for things like subcommands and flags, which are covered below.

示例

Being a programmer can be a lonely job. Thankfully by the power of automation that is not the case! Let's create a greeter app to fend off our demons of loneliness!

Start by creating a directory named `greet`, and within it, add a file, `greet.go` with the following code in it:

```
package main

import (
  "os"
  "github.com/codegangsta/cli"
)

func main() {
  app := cli.NewApp()
  app.Name = "greet"
  app.Usage = "fight the loneliness!"
  app.Action = func(c *cli.Context) {
    println("Hello friend!")
  }

  app.Run(os.Args)
}
```

Install our command to the `$GOPATH/bin` directory:

```
$ go install
```

Finally run our new command:

```
$ greet
Hello friend!
```

`cli.go` also generates neat help text:

```
$ greet help
NAME:
    greet - fight the loneliness!

USAGE:
    greet [global options] command [command options] [arguments...]

VERSION:
    0.0.0

COMMANDS:
    help, h  Shows a list of commands or help for one command

GLOBAL OPTIONS
    --version   Shows version information
```

参数

You can lookup arguments by calling the `Args` function on `cli.Context`.

```
...
app.Action = func(c *cli.Context) {
  println("Hello", c.Args()[0])
}
...
```

选项标记

Setting and querying flags is simple.

```
...
app.Flags = []cli.Flag {
  cli.StringFlag{
    Name: "lang",
    Value: "english",
    Usage: "language for the greeting",
  },
}
app.Action = func(c *cli.Context) {
  name := "someone"
  if len(c.Args()) > 0 {
    name = c.Args()[0]
  }
  if c.String("lang") == "spanish" {
    println("Hola", name)
  } else {
    println("Hello", name)
  }
}
...
```

You can also set a destination variable for a flag, to which the content will be scanned.

```
...
var language string
app.Flags = []cli.Flag {
  cli.StringFlag{
    Name:        "lang",
    Value:       "english",
    Usage:       "language for the greeting",
    Destination: &language,
  },
}
app.Action = func(c *cli.Context) {
  name := "someone"
  if len(c.Args()) > 0 {
    name = c.Args()[0]
  }
  if language == "spanish" {
    println("Hola", name)
  } else {
    println("Hello", name)
  }
}
...
```
See full list of flags at http://godoc.org/github.com/codegangsta/cli

选项别名

You can set alternate (or short) names for flags by providing a comma-delimited list for the `Name`. e.g.

```
app.Flags = []cli.Flag {
  cli.StringFlag{
    Name: "lang, l",
    Value: "english",
    Usage: "language for the greeting",
  },
}
```
That flag can then be set with `--lang spanish` or `-l spanish`. Note that giving two different forms of the same flag in the same command invocation is an error.

环境变量中取值

You can also have the default value set from the environment via `EnvVar`. e.g.

```
app.Flags = []cli.Flag {
  cli.StringFlag{
    Name: "lang, l",
    Value: "english",
    Usage: "language for the greeting",
    EnvVar: "APP_LANG",
  },
}
```
The `EnvVar` may also be given as a comma-delimited "cascade", where the first environment variable that resolves is used as the default.

```
app.Flags = []cli.Flag {
  cli.StringFlag{
    Name: "lang, l",
    Value: "english",
    Usage: "language for the greeting",
    EnvVar: "LEGACY_COMPAT_LANG,APP_LANG,LANG",
  },
}
```

子命令

Subcommands can be defined for a more git-like command line app.

```
...
app.Commands = []cli.Command{
  {
    Name:      "add",
    Aliases:     []string{"a"},
    Usage:     "add a task to the list",
    Action: func(c *cli.Context) {
      println("added task: ", c.Args().First())
    },
  },
  {
    Name:      "complete",
    Aliases:     []string{"c"},
    Usage:     "complete a task on the list",
    Action: func(c *cli.Context) {
      println("completed task: ", c.Args().First())
    },
  },
  {
    Name:      "template",
    Aliases:     []string{"r"},
    Usage:     "options for task templates",
    Subcommands: []cli.Command{
      {
        Name:  "add",
        Usage: "add a new template",
        Action: func(c *cli.Context) {
            println("new task template: ", c.Args().First())
        },
      },
      {
        Name:  "remove",
        Usage: "remove an existing template",
        Action: func(c *cli.Context) {
          println("removed task template: ", c.Args().First())
        },
      },
    },
  },
}
...
```

自动完成

You can enable completion commands by setting the `EnableBashCompletion` flag on the `App` object. By default, this setting will only auto-complete to show an app's subcommands, but you can write your own completion methods for the App or its subcommands.

```
...
var tasks = []string{"cook", "clean", "laundry", "eat", "sleep", "code"}
app := cli.NewApp()
app.EnableBashCompletion = true
app.Commands = []cli.Command{
  {
    Name:  "complete",
    Aliases: []string{"c"},
    Usage: "complete a task on the list",
    Action: func(c *cli.Context) {
       println("completed task: ", c.Args().First())
    },
    BashComplete: func(c *cli.Context) {
      // This will complete if no args are passed
      if len(c.Args()) > 0 {
        return
      }
      for _, t := range tasks {
        fmt.Println(t)
      }
    },
  }
}
...
```

启用

Source the `autocomplete/bash_autocomplete` file in your `.bashrc` file while setting the `PROG` variable to the name of your program:

`PROG=myprogram source /.../cli/autocomplete/bash_autocomplete`

安装部署

Copy `autocomplete/bash_autocomplete` into `/etc/bash_completion.d/` and rename it to the name of the program you wish to add autocomplete support for (or automatically install it there if you are distributing a package). Don't forget to source the file to make it active in the current shell.

```
sudo cp src/bash_autocomplete /etc/bash_completion.d/<myprogram>
source /etc/bash_completion.d/<myprogram>
```

Alternatively, you can just document that users should source the generic `autocomplete/bash_autocomplete` in their bash configuration with `$PROG` set to the name of their program (as above).

参与指南

Feel free to put up a pull request to fix a bug or maybe add a feature. I will give it a code review and make sure that it does not break backwards compatibility. If I or any other collaborators agree that it is in line with the vision of the project, we will work with you to get the code into a mergeable state and merge it into the master branch.

If you have contributed something significant to the project, I will most likely add you as a collaborator. As a collaborator you are given the ability to merge others pull requests. It is very important that new code does not break existing code, so be careful about what code you do choose to merge. If you have any questions feel free to link @codegangsta to the issue in question and we can review it together.

If you feel like you have contributed to the project but have not yet been added as a collaborator, I probably forgot to add you. Hit @codegangsta up over email and we will get it figured out.

https://github.com/codegangsta/cli