Go 每日一库之 go-homedir

Original dj GoUpUp 2020-01-19

收录于话题
#Go 每日一库

48个 >

简介

今天我们来看一个很小，很实用的库go-homedir。顾名思义，`go-homedir`用来获取用户的主目录。实际上，使用标准库`os/user`我们也可以得到这个信息：

```
package main

import (
  "fmt"
  "log"
  "os/user"
)

func main() {
  u, err := user.Current()
  if err != nil {
    log.Fatal(err)
  }
```

```
  fmt.Println("Home dir:", u.HomeDir)
}
```

那么为什么还要`go-homedir`库？

在 Darwin 系统上，标准库`os/user`的使用需要 cgo。所以，任何使用`os/user`的代码都不能交叉编译。但是，大多数人使用`os/user`的目的仅仅只是想获取主目录。因此，`go-homedir`库出现了。

快速使用

`go-homedir`是第三方包，使用前需要先安装：

```
$ go get github.com/mitchellh/go-homedir
```

使用非常简单：

```
package main

import (
  "fmt"
  "log"

  "github.com/mitchellh/go-homedir"
)

func main() {
  dir, err := homedir.Dir()
  if err != nil {
    log.Fatal(err)
```

```
  }

  fmt.Println("Home dir:", dir)

  dir = "~/golang/src"
  expandedDir, err := homedir.Expand(dir)
  if err != nil {
    log.Fatal(err)
  }

  fmt.Printf("Expand of %s is: %s\n", dir, expandedDir)
}
```

`go-homedir`有两个功能：

- `Dir`：获取用户主目录；
- `Expand`：将路径中的第一个`~`扩展成用户主目录。

高级用法

由于`Dir`的调用可能涉及一些系统调用和外部执行命令，多次调用费性能。所以`go-homedir`提供了缓存的功能。默认情况下，缓存是开启的。我们也可以将`DisableCache`设置为`false`来关闭它。

```
package main

import (
  "fmt"
  "log"
```

```
  "github.com/mitchellh/go-homedir"
)

func main() {
  homedir.DisableCache = false

  dir, err := homedir.Dir()
  if err != nil {
    log.Fatal(err)
  }

  fmt.Println("Home dir:", dir)
}
```

使用缓存时，如果程序运行中修改了主目录，再次调用`Dir`还是返回之前的目录。如果需要获取最新的主目录，可以先调用`Reset`清除缓存。

实现

`go-homedir`源码只有一个文件homedir.go，今天我们大概看一下`Dir`的实现，去掉缓存相关代码：

```
func Dir() (string, error) {
  var result string
  var err error
  if runtime.GOOS == "windows" {
    result, err = dirWindows()
  } else {
    // Unix-like system, so just assume Unix
    result, err = dirUnix()
  }
```

```
  if err != nil {
    return "", err
  }
  return result, nil
}
```

判断当前的系统是`windows`还是类 Unix，分别调用不同的方法。先看 windows 的，比较简单：

```
func dirWindows() (string, error) {
    // First prefer the HOME environmental variable
    if home := os.Getenv("HOME"); home != "" {
        return home, nil
    }

    // Prefer standard environment variable USERPROFILE
    if home := os.Getenv("USERPROFILE"); home != "" {
        return home, nil
    }

    drive := os.Getenv("HOMEDRIVE")
    path := os.Getenv("HOMEPATH")
    home := drive + path
    if drive == "" || path == "" {
        return "", errors.New("HOMEDRIVE, HOMEPATH, or USERPROFILE are blank")
    }

    return home, nil
}
```

流程如下：

- 读取环境变量`HOME`，如果不为空，返回这个值；
- 读取环境变量`USERPROFILE`，如果不为空，返回这个值；
- 读取环境变量`HOMEDRIVE`和`HOMEPATH`，如果两者都不为空，拼接这两个值返回。

类 Unix 系统的实现稍微复杂一点：

```
func dirUnix() (string, error) {
  homeEnv := "HOME"
  if runtime.GOOS == "plan9" {
    // On plan9, env vars are lowercase.
    homeEnv = "home"
  }

  // First prefer the HOME environmental variable
  if home := os.Getenv(homeEnv); home != "" {
    return home, nil
  }

  var stdout bytes.Buffer

  // If that fails, try OS specific commands
  if runtime.GOOS == "darwin" {
    cmd := exec.Command("sh", "-c", `dscl -q . -read /Users/"$(whoami)" NFSHomeDirectory | sed 's/^[^ ]*: //'`)
    cmd.Stdout = &stdout
    if err := cmd.Run(); err == nil {
      result := strings.TrimSpace(stdout.String())
      if result != "" {
        return result, nil
      }
```

```
    }
  } else {
    cmd := exec.Command("getent", "passwd", strconv.Itoa(os.Getuid()))
    cmd.Stdout = &stdout
    if err := cmd.Run(); err != nil {
      // If the error is ErrNotFound, we ignore it. Otherwise, return it.
      if err != exec.ErrNotFound {
        return "", err
      }
    } else {
      if passwd := strings.TrimSpace(stdout.String()); passwd != "" {
        // username:password:uid:gid:gecos:home:shell
        passwdParts := strings.SplitN(passwd, ":", 7)
        if len(passwdParts) > 5 {
          return passwdParts[5], nil
        }
      }
    }
  }

  // If all else fails, try the shell
  stdout.Reset()
  cmd := exec.Command("sh", "-c", "cd && pwd")
  cmd.Stdout = &stdout
  if err := cmd.Run(); err != nil {
    return "", err
  }

  result := strings.TrimSpace(stdout.String())
  if result == "" {
    return "", errors.New("blank output when reading home directory")
```

```
    }


    return result, nil
}
```

流程如下：

- 先读取环境变量`HOME`（注意 plan9 系统上为`home`），如果不为空，返回这个值；
- 使用`getnet`命令查看系统的数据库中的相关记录，我们知道`passwd`文件中存储了用户信息，包括用户的主目录。使用`getent`命令查看`passwd`中当前用户的那条记录，然后从中找到主目录部分返回；
- 如果上一个步骤失败了，我们知道`cd`后不加参数是直接切换到用户主目录的，而`pwd`可以显示当前目录。那么就可以结合这两个命令返回主目录。

这里分析源码并不是表示使用任何库都要熟悉它的源码，毕竟使用库就是为了方便开发。 但是源码是我们学习和提高的一个非常重要的途径。我们在使用库遇到问题的时候也要有能力从文档或甚至源码中查找原因。

参考

1. home-dir GitHub 仓库

我

我的博客

欢迎关注我的微信公众号【GoUpUp】，共同学习，一起进步~

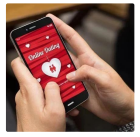
People who liked this content also liked

Go 每日一库之 reflect

GoUpUp

恋爱一年的“女友”竟是堂婶……男子崩溃了！

中国反邪教



大牌香水怎么买最便宜？我的骨灰级攻略都在这篇！

Lisa的美妆日记