View on GitHub

Sevice Computing

Principle, Technology and Architecture for building effitive, elastic and solid services on cloud

CLI 命令行实用程序开发基础

1、概述

CLI（Command Line Interface）实用程序是Linux下应用开发的基础。正确的编写命令行程序让应用与操作系统融为一体，通过shell或script使得应用获得最大的灵活性与开发效率。Linux提供了cat、ls、copy等命令与操作系统交互；go语言提供一组实用程序完成从编码、编译、库管理、产品发布全过程支持；容器服务如docker、k8s提供了大量实用程序支撑云服务的开发、部署、监控、访问等管理任务；git、npm等都是大家比较熟悉的工具。尽管操作系统与应用系统服务可视化、图形化，但在开发领域，CLI在编程、调试、运维、管理中提供了图形化程序不可替代的灵活性与效率。

2、基础知识

几乎所有语言都提供了完善的 CLI 实用程序支持工具。以下是一些入门文档（c 语言）：

- 开发 Linux 命令行实用程序。
- Linux命令行程序设计

如果你熟悉 python：

- Using Python to create UNIX command line tools

阅读以后你应该知道 POSIX/GNU 命令行接口的一些概念与规范。命令行程序主要涉及内容：

- 命令
- 命令行参数
- 选项：长格式、短格式
- IO：stdin、stdout、stderr、管道、重定向
- 环境变量

3、Golang 的支持

使用os，flag包，最简单处理参数的代码：

```
package main
```

```
import (
    "fmt"
    "os"
)

func main() {
    for i, a := range os.Args[1:] {
        fmt.Printf("Argument %d is %s\n", i+1, a)
    }

}
```

使用flag包的代码：

```
package main

import (
    "flag"
    "fmt"
)

func main() {
    var port int
    flag.IntVar(&port, "p", 8000, "specify port to use.  defaults to 8000.")
    flag.Parse()

    fmt.Printf("port = %d\n", port)
    fmt.Printf("other args: %+v\n", flag.Args())
}
```

中文参考：

- 标准库—命令行参数解析FLAG
- Go学习笔记：flag库的使用

更多代码实践：

- cat demo
- goimports 实现

4、开发实践

使用 golang 开发 开发 Linux 命令行实用程序 中的 **selpg**

提示：

1. 请按文档 **使用 selpg** 章节要求测试你的程序
2. 请使用 pflag 替代 goflag 以满足 Unix 命令行规范， 参考：Golang之使用Flag和Pflag
3. golang 文件读写、读环境变量，请自己查 os 包
4. “-dXXX” 实现，请自己查 `os/exec` 库，例如案例 Command，管理子进程的标准输入和输出通常使用 `io.Pipe`，具体案例见 Pipe

5、代码提交

- 在 Github 提交程序，并在 readme.md 文件中描述设计说明，使用与测试结果。
- 如果你写了 关于 golang 或其他与课程相关的博客，请在课程群提交记录博客

参考文献：
[1] Package flag: https://go-zh.org/pkg/flag/
[2] Package os: https://go-zh.org/pkg/os/
[3] CLI: Command Line Programming with Go

Sevice Computing maintained by pmlpml