CLI

服务计算命令行开发

17343140 杨泽涛

实验目的

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]()

实验内容

处理参数部分

根据网站要求，使用pflag来处理参数，相关代码如下

```
import flag "github.com/spf13/pflag"

type selpg_args struct {
    start_page int
    end_page int
    in_filename string
    page_len int
    page_type bool
    print_dest string
}
```

```
var sa selpg_args;
flag.IntVarP(&(sa.start_page), "start_page", "s",-1, "specify page of start.  defaults to -1.")
flag.IntVarP(&(sa.end_page),"end_page","e",-1,"specify page of end. default to -1.")
flag.IntVarP(&(sa.page_len),"page_len","l",72,"specify length of page, default to 72.")
flag.BoolVarP(&(sa.page_type),"page_type","f",false,"specify type of page, default to false.")
flag.StringVarP(&(sa.print_dest),"print_dest","d","","specify print_dest if needed.")
flag.Parse()
```

根据selpg文档要求，我们设置参数报错程序如下

```go
func check(sa selpg_args){
    if (sa.start_page < 0){
        fmt.Fprintf(os.Stderr, "start_page should be specified correctly!\n")
        os.Exit(1)
    }
    if (sa.end_page < 0){
        fmt.Fprintf(os.Stderr, "end_page should be specified correctly!\n")
        os.Exit(1)
    }
    if(sa.start_page>sa.end_page){
        fmt.Fprintf(os.Stderr, "start_page should be less than end_page!\n")
        os.Exit(1)
    }
    if(sa.page_type == true) && (sa.page_len!=72){
        fmt.Fprintf(os.Stderr, "-l and -lf is exclusive\n")
        os.Exit(1)
    }
}
```

读写创建部分

我们根据参数的类型来为我们的输入输出创建正确的对象：

```go
if(sa.in_filename!=""){
     _, errtest := os.Stat(sa.in_filename)
    if os.IsNotExist(errtest) {
        fmt.Fprintf(os.Stderr, "Open File Error\n")
        os.Exit(1)
    }
}

var myin *os.File
if sa.in_filename==""{
    myin=os.Stdin;
} else{
    myin, _ = os.Open(sa.in_filename)
}

var myout io.WriteCloser
var err error
if len(sa.print_dest) == 0 {
    myout = os.Stdout
} else {
    //本段代码借鉴了网上，因为我不了解管道的相关API
    cmd := exec.Command("lp", "-d"+sa.print_dest)
    cmd.Stdout, err = os.OpenFile(sa.print_dest, os.O_APPEND|os.O_WRONLY, os.ModeAppend)
    myout, err = cmd.StdinPipe()
    if err != nil {
        fmt.Fprintf(os.Stderr, "-d pipe error! \n")
        os.Exit(1)
    }
    cmd.Run()
}
```

程序读写部分

我们根据定长和不定长来进行不同类型的读写操作

```go
lineCounter:=0
pageCounter:=1
var mybuf string
buf := bufio.NewReader(myin)
if sa.page_type==false{
    for true{
            mybuf, err = buf.ReadString('\n')
            lineCounter++
            if lineCounter > sa.page_len {
                pageCounter++
                lineCounter = 1
            }
            if (pageCounter < sa.start_page) || (pageCounter > sa.end_page) {
                break
            }
            if err!=nil{
                break
            }
            _, myerror := myout.Write([]byte(mybuf))
            if myerror != nil {
                break
            }
    }
} else if sa.page_type==true{
    for true{
        mybuf, err = buf.ReadString('\f')
        pageCounter++
        if (pageCounter < sa.start_page) || (pageCounter > sa.end_page) {
                break
            }
        if err!=nil{
            break
        }
        _, myerror:=myout.Write([]byte(mybuf))
        if myerror != nil{
            break
        }
    }
}

if(pageCounter!=sa.end_page-sa.start_page+1){
    fmt.Fprintf(os.Stderr, "page number error! \n")
    os.Exit(1)
}
```

相关测试

1. `selpg -s1 -e1 input_file`

```
[root@localhost ~]# ./selpg -s1 -e1 in.txt
asdasd
sadasz
zxc123
123
%$^$%^
[root@localhost ~]#
```

2. `selpg -s1 -e1 < input_file`

```
[root@localhost ~]# ./selpg -s1 -e1 < in.txt
asdasd
sadasz
zxc123
123
%$^$%^
[root@localhost ~]#
```

3. `other_command | selpg -s10 -e20`

改成了-s1 -e1 因为小文件方便修改

```
[root@localhost ~]# cat in.txt | ./selpg -s1 -e1
asdasd
sadasz
zxc123
123
%$^$%^
[root@localhost ~]#
```

4. `selpg -s10 -e20 input_file >output_file`

```
[root@localhost ~]# ./selpg -s1 -e1 in.txt > out.txt
[root@localhost ~]#
```

selpg.go | out.txt

```
asdasd
sadasz
zxc123
123
%$^$%^
```

5. `$ selpg -s10 -e20 input_file 2>error_file`

```
[root@localhost ~]# ./selpg -s1 -e0 in.txt 2>err.txt
[root@localhost ~]#
```

selpg.go × out.txt × err.txt ×

```
start_page should be less than end_page!
```

6. `selpg -s10 -e20 input_file >output_file 2>error_file`

```
[root@localhost ~]# ./selpg -s1 -e0 in.txt >out.txt 2>err.txt
[root@localhost ~]#
```

selpg.go × out.txt × err.txt ×

```
start_page should be less than end_page!
```

7. `selpg -s10 -e20 input_file >output_file 2>/dev/null`

```
[root@localhost ~]# ./selpg -s1 -e1 in.txt >out.txt 2>/dev/null
[root@localhost ~]#
```

8. `$ selpg -s10 -e20 input_file >/dev/null`

```
[root@localhost ~]# ./selpg -s1 -e1 in.txt >/dev/null
[root@localhost ~]#
```

9. `selpg -s10 -e20 input_file | other_command`

```
[root@localhost ~]# ./selpg -s1 -e1 in.txt | sort
asdaddddd.txt
asdasd
hgmj
xcvxcv
[root@localhost ~]#
```

10 `selpg -s10 -e20 input_file 2>error_file | other_command`

效果和之前类似，不再重复。

11. `selpg -s10 -e20 input_file > output_file 2>error_file &`

```
[root@localhost ~]# ./selpg -s1 -e1 in.txt > out.txt 2>err.txt &
[1] 106297
[1]+  Done                    ./selpg -s1 -e1 in.txt > out.txt 2> err.txt
[root@localhost ~]#
```

测试中关于-f和-d的命令我没有测试，因为没有终端和有换页符的文件，测试中大多使用的是小文件，大文件检测在下面

```
[root@localhost ~]# ./selpg -s1 -e2 in.txt > out.txt
[root@localhost ~]#
```

in.txt行数

```
asdasd
qweqwes
safxcv
asd
asd
asdasd
xcvdsfsdf
```

Plain Text Tab Width: 8 Ln 313, Col 10

out.txt行数

```
asdasd
xcvdsfsdf
asdasd
asdasdasd
```

Plain Text Tab Width: 8 Ln 144, Col 10

指定一页12行后

```
[root@localhost ~]# ./selpg -s1 -e2 in.txt > out.txt
[root@localhost ~]# ./selpg -s1 -e2 -l12 in.txt > out.txt
[root@localhost ~]#
```

```
asd
asd
asdasd
```

Plain Text Tab Width: 8 Ln 24, Col 7