

(2)编写好脚本后,增加 shell 脚本文件的可执行权限。可以使用 `chmod` 命令增加 `add.sh` 文件的可执行权限。

```
$chmod u+x add.sh
```

(3) 在 shell 中运行以下脚本:

```
$./add.sh
1
3
6
10
15
done the sum : 15
```

下面实例修改了程序清单 28-11 中的交互式菜单, 将该菜单变成一个可以接受用户多次输入的脚本。每次处理用户的输入, 当用户输入 3 或者快捷键 E (e) 时, 循环结束, 脚本结束运行。其执行流程如图 28-5 所示。

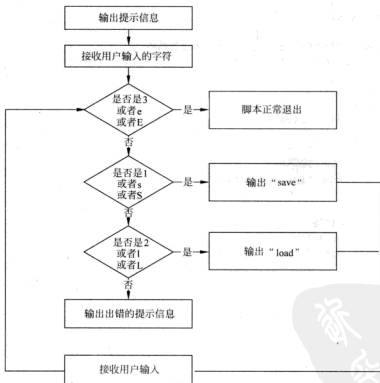


图 28-5 多次接收用户输入的交互式菜单

(1) 在 vi 编辑器中编辑以下脚本:

程序清单 28-12 mul_menu.sh 使用 while 结构语句实现一个可以多次接收输入的菜单

```
#!/bin/sh
# mul_menu.sh 使用 while 结构语句实现一个可以多次接收输入的菜单

echo "1 save"
echo "2 load"
echo "3 exit"
echo    # 输出一个换行

# 接收用户输入的内容
echo "please choose"
read chioce

# 使用 while 循环结构实现多次接收用户的输入

# 快捷键如下
# s——存储 (save)
# l——加载 (load)
# e——退出 (exit)

# 循环结构, 当用输入的不是 3、E 和 e 的时候进行循环
while [ $chioce != "3" -a $chioce != "E" -a $chioce != "e" ]
do
    case $chioce in
        1 | S | s)
            echo "save";;
        2 | L | l)
            echo "load";;
        *) # 其他的输入情况
            echo "invalid choice";;
    esac

    # 再次接收用户的输入
    echo "please choose"
    read chioce
done

echo "done"

exit 0
```

(2) 编写好脚本后, 增加 shell 脚本文件的可执行权限。可以使用 `chmod` 命令增加 `mul_menu.sh` 文件的可执行权限。

```
$chmod u+x mul_menu.sh
```

(3) 在 shell 中运行以下脚本:

```
$/mul_menu.sh
1 save
```

```
2 load
3 exit
please choose
```

(4) 输入 1。

```
1
save
please choose
```

(5) 输入字符 a。

```
a
invalid choice
please choose
```

(6) 输入大写字母 E。

```
E
done
```

28.4.2 until 循环结构

until 循环同样可以实现循环结构，其形式如下：

```
until 条件判断命令
do
    命令
done
```

until 循环和 while 循环不同。while 循环中的条件判断命令表示的是执行循环体命令的条件，而 until 循环中的条件判断命令表示的是结束循环体命令执行的条件。其执行流程如图 28-6 所示。

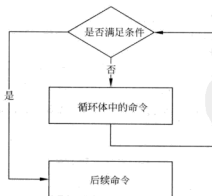


图 28-6 until 循环的执行流程

下面实例演示了使用 until 循环结构计算 1~5 的整数的和。

(1) 在 vi 编辑器中编辑以下脚本：

程序清单 28-13 until_add.sh 使用 until 结构计算 1~5 的整数的和

```
#!/bin/sh
# until_add.sh 使用 until 结构计算 1~5 的整数的和

i=1

# 条件判断命令表示的是循环结束的条件
until [ $i -gt 5 ]
do
    sum='expr $sum + $i' # 累加和
    echo $sum
    i='expr $i + 1'
done

echo "done the sum : $sum"

exit 0
```

(2) 编写好脚本后, 增加 shell 脚本文件的可执行权限。可以使用 chmod 命令增加 until_add.sh 文件的可执行权限。

```
$chmod u+x until_add.sh
```

(3) 在 shell 中运行以下脚本:

```
$/until_add.sh
1
3
6
10
15
done the sum : 15
```

下面实例使用 until 重新实现交互式菜单, 需要改变的只是循环的条件判断命令。

(1) 在 vi 编辑器中编辑以下脚本:

程序清单 28-14 until_menu.sh 使用 until 结构语句实现一个可以多次接收输入的菜单

```
#!/bin/sh
# until_menu.sh 使用 until 结构语句实现一个可以多次接收输入的菜单

echo "1 save"
echo "2 load"
echo "3 exit"
echo    # 输出一个换行

# 接收用户输入的内容
echo "please choose"
read chioce
```

```

# 使用 until 循环结构实现多次接收用户的输入

# 快捷键如下
# s——存储 (save)
# l——加载 (load)
# e——退出 (exit)

# 循环结构，当用户输入的不是 3、E 和 e 的时候进行循环
# 注意，由于使用的是 until 循环，需要改变条件判断命令的条件

until [ $chioce = "3" -o $chioce = "E" -o $chioce = "e" ]
do
    case $chioce in
        1 | S | s)
            echo "save";;
        2 | L | l)
            echo "load";;
        *) # 其他的输入情况
            echo "invalid choice";;
    esac

    # 再次接收用户的输入
    echo "please choose"
    read chioce
done

echo "done"

exit 0

```

(2) 编写好脚本后，增加 shell 脚本文件的可执行权限。可以使用 `chmod` 命令增加 `until_menu.sh` 文件的可执行权限。

```
$chmod u+x until_menu.sh
```

(3) 在 shell 中运行以下脚本：

```

$./until_menu.sh
1 save
2 load
3 exit
please choose

```

(4) 输入小写字母 `s`。

```

s
save
please choose

```

(5) 输入小写字母 `e`。

```
e
done
```

28.4.3 for 循环结构

for 循环实现循环结构，其形式如下：

```
for 循环因子变量 in 循环参数列表
do
    命令
done
```

for 循环和 C 语言中的 for 循环有所不同。shell 脚本中的 for 循环依次处理循环参数列表中的每个参数，将这个参数的值赋值给循环因子变量。当循环参数列表中参数处理完毕后，for 循环就停止执行了。其执行流程如图 28-7 所示。

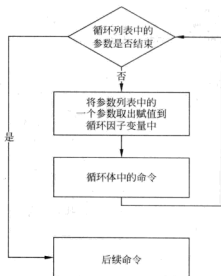


图 28-7 for 循环的执行流程

下面实例演示一个简单的 for 循环，计算 1~5 的整数的和。

(1) 在 vi 编辑器中编辑以下脚本：

程序清单 28-15 for_add.sh 计算 1~5 的整数的和

```
#!/bin/sh
# for_add.sh 计算 1~5 的整数的和

#使用 for 循环实现累加，循环变量 i 依次等于循环参数列表中的值
for i in 1 2 3 4 5
do
    sum='expr $sum + $i'
    echo "i=$i"
```

```

    echo "sum=$sum"
done

echo "done ths sum : $sum"

exit 0

```

(2) 编写好脚本后, 增加 shell 脚本文件的可执行权限。可以使用 `chmod` 命令增加 `for_add.sh` 文件的可执行权限。

```

$chmod u+x for_add.sh

```

(3) 在 shell 中运行以下脚本:

```

$./ for_add.sh
i=1
sum=1
i=2
sum=3
i=3
sum=6
i=4
sum=10
i=5
sum=15
done ths sum : 15

```

下面实例演示对当前目录下的所有文件进行类型判断的操作。该脚本使用一个 `forx` 循环来实现, 当前目录下所有的文件名作为循环参数列表, 处理每一个文件名, 使用 `if/else` 分支结构对其进行类型判断, 本实例只判断普通文件和目录文件两种最常见的文件类型。

(1) 在 vi 编辑器中编辑以下脚本:

程序清单 28-16 list.sh 判断当前目录下每个文件的类型

```

#!/bin/sh
# list.sh 判断当前目录下每个文件的类型

# 使用 for 循环, 循环参数列表是使用 ls 命令的输出结果
for file in `ls`
do
    if [ -f $file ]      # 判断普通文件
    then
        echo "$file is a regular file"
    elif [ -d $file ]    # 判断目录文件
    then
        echo "$file is a directory"
    else                # 判断其他文件
        echo "unknown file type"
    fi
done

```

```
exit 0
```

(2) 编写好脚本后, 增加 shell 脚本文件的可执行权限。可以使用 `chmod` 命令增加 `list.sh` 文件的可执行权限。

```
$chmod u+x list.sh
```

(3) 列出当前目录下的文件与目录如下:

```
$ls -l
-rw-r--r--  1 root    0 Feb  7  04:50 test1.txt
-rw-r--r--  1 root    0 Feb  7  04:51 test2.txt
drw-r--r--  1 root   56 Feb  7  04:52 dir
lrw-r--r--  1 root   45 Feb  7  04:53 link
```

(4) 在 shell 中运行以下脚本:

```
$/list.sh
test1.txt is a regular file
test2.txt is a regular file
dir is a directory
link unknown type
```

28.4.4 break 和 continue 命令

shell 脚本中同样支持对循环内部的跳转控制——`break` 语句立即退出循环, 而 `continue` 语句忽略本循环中的其他命令, 继续下一次循环。也就是说 `break` 语句跳出当前循环, 而 `continue` 语句跳到循环的起始位置。这一点和 C 语言一样。`break` 语句和 `continue` 语句的执行流程如图 28-8 所示。

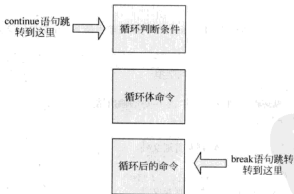


图 28-8 break 语句和 continue 语句的执行流程

在编写 shell 脚本的过程中, 有时用户需要用到无限循环的技巧, 也就是说这个循环一直执行, 是一个死循环。Linux 系统中的内置命令的 `true` 总是返回 0 值 (真), 而内置命令 `false` 则返回非零值 (假)。下面是两种编写死循环的形式。


```
#一直执行到程序执行了 break 或用户强行中断时才结束循环
while true
do
    命令
done
```

上面所示的循环也可以使用 `until false` 的循环结构来表示。

```
#一直执行到程序执行了 break 或用户强行中断时才结束循环
until false
do
    命令
done
```

下面实例使用 `break` 语句改写之前的交互式菜单。当用户输入 3、E 或者 e 的时候会跳出死循环。程序的执行流程如图 28-9 所示。

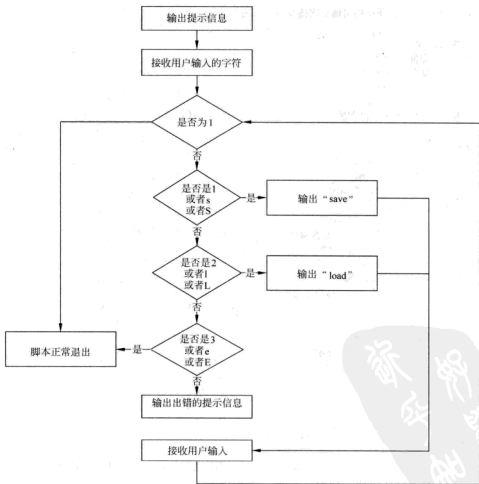


图 28-9 交互式菜单的执行流程

(1) 在 vi 编辑器中编辑以下脚本:

程序清单 28-17 final_menu.sh 使用 break 语句和死循环实现一个可以多次接收输入的菜单

```
#!/bin/sh
# final_menu.sh 使用 break 语句和死循环实现一个可以多次接收输入的菜单

echo "1 save"
echo "2 load"
echo "3 exit"
echo    # 输出一个换行

# 接收用户输入的内容
echo "please choose"
read chioce

# 使用 while 循环结构实现多次接收用户的输入

# 快捷键如下
# s——存储 (save)
# l——加载 (load)
# e——退出 (exit)

# 循环结构, 当用户输入的不是 3、E 和 e 的时候进行循环
while true
do
    case $chioce in
        1 | S | s)
            echo "save";;
        2 | L | l)
            echo "load";;
        3 | E | e) # 输入退出命令时使用 break 语句跳出循环
            echo "exit"
            break;;
        *) # 其他的输入情况
            echo "invalid choice";;
    esac

    # 再次接收用户的输入
    echo "please choose"
    read chioce
done

echo "done"

exit 0
```

(2) 编写好脚本后, 增加 shell 脚本文件的可执行权限。可以使用 chmod 命令增加 final_menu.sh 文件的可执行权限。

```
$chmod u+x final_menu.sh
```

(3) 在 shell 中运行以下脚本:

```
$/final_menu.sh
1 save
2 load
3 exit
please choose
```


(4) 输入大写字母 E。

```
E
exit
done
```

28.5 定义函数和使用函数

同 C 语言一样, shell 脚本也提供了函数功能。函数通常也称之为子过程。shell 中的函数的定义格式如下所示:

```
函数名()
{
    函数中的命令
    ; # 注意这里有一个分号
}
```

 **注意:** 当定义一个函数的时候,除了要在两个“{}”内部书写函数中的命令外,不要忘记在函数的结尾位置添加一个“;”。

这样 shell 才能够识别一个函数的命令结束了。使用函数的一个好处就是使程序更加的结构化和模块化。当定义一个函数后,用户就可以在 shell 脚本中对定义好的函数进行调用,这样就可以将一个复杂的 shell 脚本分为若干个方便管理的 shell 脚本段,如下所示:

```
# shell 脚本起始

# initial 函数, 负责初始化
initial()
{
    命令列表
    ;
}

# handle 函数, shell 脚本的主题, 处理数据, 执行 shell 脚本的指令
handle()
{
    命令列表
    ;
}
```

```

# cleanup 函数, 做一些清理工作
cleanup()
{
    命令列表
;
}

# error 函数, 负责处理 shell 脚本执行过程中出现的错误
error()
{
    命令列表
;
}

# shell 脚本的执行体, 相当于 C 语言中的 main 函数

# 调用初始化函数, 将程序初始化
echo "initializing..."
initial # 在 shell 脚本中调用函数只需要写函数名就可以了, 不需要参数

# 调用 handle 函数, 处理数据, 执行命令。如果出现错误, 则调用 error 函数处理
echo "handling..."
handle

# 调用 cleanup 函数, 做清理工作
echo "cleaning..."
cleanup

# shell 脚本结束

```

shell 中的函数没有参数, 但是可以有返回值。

 **注意:** 对于 shell 脚本来讲每一个返回值的类型都是字符串。

下面实例演示了使用函数封装接收用户输入的操作。修改之前的菜单脚本, 在 input 函数内部接收用户的输入。

(1) 在 vi 编辑器中编辑以下脚本:

程序清单 28-18 func_menu.sh 使用 input 函数封装用户的输入

```

#!/bin/sh
# func_menu.sh 使用 input 函数封装用户的输入

input()
{
    read choice

    case $choice in

```

```

1 | S | s) # 保存命令
    return 1;;
2 | L | l) # 加载命令
    return 2;;
3 | E | e) # 退出命令
    return 3;;
*) # 其他的输入情况
    return -1;;
; # 不要忘记这个分号
}

echo "1 save"
echo "2 load"
echo "3 exit"
echo # 输出一个换行

# 使用 while 循环结构实现多次接收用户的输入

# 快捷键如下
# s——存储 (save)
# l——加载 (load)
# e——退出 (exit)

# 循环结构, 当用户输入的不是 3、E 和 e 的时候进行循环
while true
do

    # 接收用户输入的内容
    echo "please choose"

    case input in
        1)
            echo "save";;
        2)
            echo "load";;
        3) # 输入退出命令时使用 break 语句跳出循环
            echo "exit"
            break;;
        -1) # 其他的输入情况
            echo "invalid choice";;
    esac
done

echo "done"

exit 0

```

(2) 编写好脚本后, 增加 shell 脚本文件的可执行权限。可以使用 `chmod` 命令增加 `func_menu.sh` 文件的可执行权限。

```
$chmod u+x func_menu.sh
```

(3) 在 shell 中运行以下脚本:

```
$/func_menu.sh  
1 save  
2 load  
3 exit  
please choose
```

(4) 输入小写的字符 e。

```
e  
exit  
done
```

一线开发人员全力打造，分享技术盛宴！

重点内容及特色

- ◎ 本书全面介绍了Linux平台下C程序设计的开发环境、开发工具和典型应用，基本涵盖了Linux C程序设计的大部分内容，尤其重点对C语言的本质进行了剖析。
- ◎ 本书详细介绍了同类书中较少涉及或者讲解不透彻的很多内容，如Linux环境下的开发工具、C语言的本质、C语言中的陷阱、makefile、多线程编程、网络编程、新版内核的文件系统机制、shell脚本与C语言配合使用等。
- ◎ 本书对所有知识点均按照先阐述概念，再分析应用，最后列举实例的方式进行讲解，可以让读者比较容易地理解Linux环境编程的要点。
- ◎ 本书使用大量示例进行讲解，可以让读者更能体会到所学知识在实际开发中的应用。
- ◎ 和国外的一些经典图书相比，本书列举了大量的典型实例，具有超强的实用性。而且本书讲解更加贴近中国人的阅读习惯，理解起来更加容易。

读者对象

- ◎ 想要全面学习Linux C编程的人员
- ◎ 想要更加深入理解Linux C编程本质的人员
- ◎ 相关培训班的培训学员和老师
- ◎ 需要一本案头必备查询手册的程序员
- ◎ 其他Linux C程序设计爱好者

特别提示

- ◎ 本书所有源代码请到清华大学出版社的网站（www.tup.com.cn）上下载。请先在主页上的搜索栏中输入书名搜索到本书信息，然后找到下载信息下载即可。

