第 5 章

Shell 脚本程序设计

从本章开始，我们将把重点转到 Shell 编程上。实际上，读者很可能已经不止一次地接触到了 Shell，只是没有注意到而已：当你使用 SSH 客户端工具远程连接到系统，或坐在一台服务器前输入密码登录到系统中时，面前呈现的跳动的光标就是一个 Shell。在计算机语言中，Shell 指一种命令行解释器，是为用户和操作系统进行通信提供的一种接口（想象一下，如果没有一种与计算机沟通的方式，那么计算机如何得到来自人脑的指令呢），它接收来自用户输入的命令，并将其转换为一系列的系统调用送到内核执行，并将结果输出给用户。

5.1 初识 Shell

在学习任何一门编程语言时，都会对其编译工具进行一定的了解。本节将介绍 Shell 和 Shell 脚本的区别以及运行第一个 Shell 脚本，并对 Shell 的缺陷进行介绍。

1. Shell 和 Shell 脚本

Shell 指一种应用程序，这个应用程序提供了一个界面，用户通过这个界面访问操作系统内核的服务。Ken Thompson 的 Shell 是第一种 UNIX Shell，Windows Explorer 是一个典型的图形界面 Shell。

Shell 脚本（Shell script），是一种为 Shell 编写的脚本程序。业界所说的 Shell 通常都指 Shell 脚本。Shell 和 Shell script 是两个不同的概念。一般提到的“Shell 编程”都指 Shell 脚本编程，不是指开发 Shell 自身（如 Windows Explorer 扩展开发）。

整个 Linux 系统都是用 Shell 脚本串联起来的，如 vim /etc/rc.d/rc.sysinit，甚至很多 Linux 内置命令本身就是 Shell，如 services 命令、startx 命令等。因此，掌握 Shell 是掌握 Linux 源码结构和进行 Linux 编程的基础。

2. 运行 Shell 脚本

在终端输入以下命令，新建一个名为 jxnu.sh 的文件（文件的扩展名为 sh，sh 代表 Shell，扩展名不会影响脚本的运行，只是起到见名知义的作用）。

```
[root@localhost ~]#vim jxnu.sh
```

进入 vim 编辑器后，输入以下命令，就完成了本书的第一个 Shell 脚本（其中“#!”是一个约定的标记，用来指定解析器。echo 命令用于向窗口输出文本）。

```
#!/bin/bash
echo "hello world!"
```

因为 Shell 程序写完后并没有执行权限，所以要想执行该脚本，可以使用 chmod 命令赋予该脚本可执行的权限，然后再执行。

```
[root@localhost ~]#chmod +x jxnu.sh
[root@localhost ~]#./jxnu.sh
```

有 3 种方式可以执行 Shell 脚本。

（1）使用./filename.sh 运行当前目录下的 Shell 脚本。

```
[root@localhost ~]#./jxnu.sh
```

执行结束后的运行结果如下。

```
hello world!
```

（2）调用解释器使得脚本执行，如 bash、csh、ash、bsh、ksh 等。

```
[root@localhost ~]#sh jxnu.sh
```

执行结束后的运行结果如下。

```
hello world!
```

（3）使用 source 命令。

```
[root@localhost ~]#source jxnu.sh
```

执行结束后的运行结果如下。

```
hello world!
```

3. Shell 的缺陷

Shell 只定义了一个非常简单的编程语言，所以如果脚本程序复杂度较高，或者要操作的数据结构比较复杂，那么还是应该使用 Python、Perl 这样的脚本语言，或者是你擅长的高级语言，因为 Shell 和 bash 在这方面存在以下缺陷：

（1）它的函数只能返回字符串，无法返回数组；

（2）它不支持面向对象，无法实现一些优雅的设计模式；

（3）它是解释型的，一边解释一边执行，连 PHP 那种预编译都不是，如果脚本包含错误代码（如调用了不存在的函数），只要未执行到这一行，程序就不会报错。

5.2 Shell 的输入输出与注释

1. 输入命令

read 命令用于从标准输入读取单行数据，该命令可以读取键盘输入，当使用重定向时，可以读取文件的一行数据，read 命令常用参数及说明如表 5.1 所示。

表 5.1 read 命令常用参数及说明

参 数	说 明
-p	后接提示信息，即在输入前向用户输出提示信息
-s	安静模式，不显示输入的字符，如 login 输入密码
-n/N	后接整数，规定最大输入文本的长度
-e	输入时可使用命令补全功能
-d	后接一个标志符，作为输入结束的标志，实际上当第一个字符匹配时即结束输入
-a	后接一个变量，该变量会被当成数组，然后为其赋值，默认以空格作为分隔符
-t	后接秒数，规定输入等待的时间，超出时间将终止输入，防止长时间等待
-r	屏蔽“\”，如果没有该选项，则“\”作为一个转义字符，如果有，则“\”作为正常字符

应用如例 5.1 所示。

```
[root@localhost /]# read -n 4 -p "please enter your name:" name    #输出提示，限定最
#大长度为 4
please enter your name:linu                          #当输完 4 个字符后，会自动退出
[root@localhost /]# read -s -p "please enter your password:" password
please enter your password:                          #输入 123456，此时不会显示
[root@localhost /]# echo $password
123456
[root@localhost /]# read -a array      #输入字符，用空格隔开，以数组的形式赋值给 array
a b c d e
[root@localhost /]# echo ${array[3]}
d
[root@localhost /]# read -e -p "Please enter file name:" str
                                                     #使用 -e 参数，当输入字符 b，按

Please enter file name:b                             #Tab 键就会输出以 b 开头的文件名
bin/ boot/
Please enter file name:b
[root@localhost /]# read -d h -p "Please enter file name:" str
                                                     #使用-d 参数，当输入字符 h 时自动

Please enter file name:test.sh                       #退出
[root@localhost /]#
[root@localhost /]# read -t 5 -p "Please enter file name:" str
                                                     #使用-t 参数，当超过 5s 时自动退出
```

```
Please enter file name:
[root@localhost /]#
```

例 5.1　read 命令

2. 输出命令

Shell 中可以使用 **echo 命令**和 **printf 命令**输出字符，其中 echo 在输出时会自动换行，而 printf 换行时需要手动添加换行符\n，printf 使用引用文本或空格分隔的参数，外面可以在 printf 中使用格式化字符串，还可以指定字符串的宽度、左右对齐方式等，相较于 echo 更加灵活，其语法和应用举例分别如下。

echo 语法：

```
echo 字符串
```

应用如例 5.2 所示。

```
[root@localhost /]#echo "Hello world"              #显示普通字符串
Hello world
[root@localhost /]#echo "\"Hello world\""          #显示转义字符
"Hello world"
[root@localhost /]#read name                       #显示变量
Tom
[root@localhost /]#echo "Hello $name"
Hello Tom
[root@localhost /]#echo -e "Hello\n world"         #显示换行,-e 开启转义
Hello
world
[root@localhost /]#echo "Hello world" >tmpfile     #将输出定向至 tmpfile 文件
```

例 5.2　echo 命令

printf 语法：

```
printf 格式控制字符串 [参数列表]
```

在 vim 编辑器中输入以下命令，如例 5.3 所示。

```
1  #! /bin/bash
2  printf "%-10s %-10s %-8s\n"      姓名      年龄      体重(kg)
3  printf "%-10s %-10d %.2f\n"      张三      19        56.2456
4  printf "%-10s %-10d %.2f\n"      李四      22        42.8798
5  printf "%-10s %-10d %.2f\n"      王五      20        66.2435
姓名      年龄      体重(kg)
张三      19        56.25
李四      22        42.88
王五      20        66.24
```

例 5.3　printf 命令

第 1 行：指定脚本解析器，这里是用/bin/bash 作为解析器的。

第 2 行：打印表头，其中%s、%d、%f 都是格式替代符，%s 表示输出一个字符串，%d 输出整型，%f 以小数形式输出实数，%c 表示输出一个字符。

第 3 行：打印张三的姓名、年龄、体重，%-10s 指宽度为 10 个字符，若不足 10 个字符，则以空格填充，超过则全部显示，-表示左对齐，默认右对齐，.2f 表示保留 2 位小数。

第 4 行：与第 3 行同理。

第 5 行：与第 3 行同理。

在写代码的过程中会经常需要注释代码，注释就是对代码的解释和说明，其目的是让人们能够更加轻松地理解代码，提高程序代码的可读性。养成编写注释的好习惯对未来的职业生涯有极大的好处。

1）单行注释

与之前学过的语言类似，单行代码用“#”注释即可。只需要在需要注释的代码前加上一个“#”，这一行代码便被注释了，注释的代码会被解析器自动忽略。

2）多行注释

如果在开发过程中遇到大段的代码需要临时注释，随后又要取消注释，怎么办呢？每行加一个“#”符号太费力了，可以把这一段要注释的代码用一对花括号“{}”括起来，定义成一个函数，没有地方调用这个函数，这块代码就不会执行，就达到了和注释一样的效果。

5.3 Shell 运算符

Shell 脚本编程支持多种运算符，本节主要介绍算术运算符、关系运算符、布尔运算符、逻辑运算符和文件测试运算符。

1. 算术运算符

算术运算符主要用于计算和判断，通常用条件表达式进行计算，条件表达式要放在方括号中，并且要有空格。算术运算符的表示及作用如表 5.2 所示。

表 5.2 算术运算符的表示及作用

运 算 符	作 用
+	加法运算
-	减法运算
*	乘法运算
/	除法运算
=	赋值运算
%	取余运算
==	比较两个数是否相等。相等则返回值为 0，不相等则返回值为 1(0 代表 true，1 代表 false)
!=	比较两个数是否不相等。不相等则返回值为 0，相等则返回值为 1

应用如例 5.4 所示。

```
[root@localhost /]#x=100
[root@localhost /]#y=10
[root@localhost /]#echo "sum is:$[ $x+$y ]"                    #求和
sum is:110
[root@localhost /]#echo "difference is:$[ $x-$y ]"             #求差
difference is:90
[root@localhost /]#echo "product is:$[ $x*$y ]"                #求积
product is:1000
[root@localhost /]#echo "quotient is:$[ $x/$y ]"               #求商
quotient is:10
[root@localhost /]#echo "reminder is:$[ $x%$y ]"               #取余
reminder is:0
[root@localhost /]#echo "true or false:$[ $x==$y ]"           #判断是否相等
true or false:1
[root@localhost /]#echo "true or false:$[ $x!=$y ]"           #判断是否不相等
true or false:0
```

例 5.4　算术运算符

2. 关系运算符

关系运算符用于比较两个数的大小，关系运算符只支持数字之间的比较，不支持字符串之间的比较，除非字符串的值是数字。关系运算符的表示及作用如表 5.3 所示。

表 5.3　关系运算符的表示及作用

运　算　符	作　　用
-gt(greater than)	判断左边的数是否大于右边的数。如果是，返回值为 0；否则，返回值为 1
-lt(less　than)	判断左边的数是否小于右边的数。如果是，返回值为 0；否则，返回值为 1
-eq(equal)	判断左边的数是否等于右边的数。如果是，返回值为 0；否则，返回值为 1
-ne(not equal)	判断左边的数是否不等于右边的数。如果是，返回值为 0，否则，返回值为 1
-ge(greater equal)	判断左边的数是否大于或等于右边的数。如果是，返回值为 0，否则，返回值为 1
-le(less equal)	判断左边的数是否小于或等于右边的数。如果是，返回值为 0，否则，返回值为 1

应用如例 5.5 所示。

```
[root@localhost /]#x=10 y=100
[root@localhost /]#[ $x -gt $y ]          #判断 x 是否大于 y
[root@localhost /]#echo $?
1
[root@localhost /]#[ $x -lt $y ]          #判断 x 是否小于 y
[root@localhost /]#echo $?
0
[root@localhost /]#[ $x -eq $y ]          #判断 x 是否等于 y
[root@localhost /]#echo $?
1
[root@localhost /]#[ $x -ne $y ]          #判断 x 是否不等于 y
[root@localhost /]#echo $?
0
[root@localhost /]#[ $x -ge $y ]          #判断 x 是否大于或等于 y
```

```
[root@localhost /]#echo $?
1
[root@localhost /]#[ $x -le $y ]                    #判断 x 是否小于或等于 y
[root@localhost /]#echo $?
0
```

例 5.5 关系运算符命令

3. 布尔运算符

常用的布尔运算符有与运算、或运算和非运算。布尔运算符的表示及作用如表 5.4 所示。

表 5.4 布尔运算符的表示及作用

运 算 符	作 用
-a(and)	与运算。两个表达式均为真时返回值为 0
-o(or)	或运算。只要一个表达式为真,返回值为 0
!	非运算。表达式为假时返回值为 0

应用如例 5.6 所示。

```
[root@localhost /]#x=10 y=20
[root@localhost /]#a=1 b=2
[root@localhost /]#[ $x -gt $a -a $y -gt $b ]
[root@localhost /]#echo $?
0
[root@localhost /]#[ $x -gt $a -o $y -gt $b ]
[root@localhost /]#echo $?
0
```

例 5.6 布尔运算符

4. 逻辑运算符

逻辑运算符的表示及作用如表 5.5 所示。

表 5.5 逻辑运算符的表示及作用

运 算 符	作 用
&&	逻辑上的“和”运算
||	逻辑上的“或”运算

应用如例 5.7 所示。

```
[root@localhost ~]#x=100
[root@localhost ~]#y=10
[root@localhost ~]#[ $x -gt $y ] && echo "x is greater than y"
x is greater than y
[root@localhost ~]#[ $x -lt $y ] || echo "x is greater than y"
x is greater than y
```

例 5.7 逻辑运算符

5. 文件测试运算符

文件测试运算符用于判断 Linux 文件的各种属性。文件测试运算符的表示及作用如表 5.6 所示。

表 5.6　文件测试运算符的表示及作用

运　算　符	作　　用
-b 文件名　(block)	判断文件是不是块设备文件，如果是，返回值为 0
-d 文件名　(directory)	判断文件是不是目录，如果是，返回值为 0
-f 文件名　(file)	判断文件是不是普通文件，如果是，返回值为 0
-r 文件名　(readable)	判断文件是否可读，如果是，返回值为 0
-w 文件名　(writable)	判断文件是否可写，如果是，返回值为 0
-x 文件名　(executable)	判断文件是否可执行，如果是，返回值为 0
-s 文件名　(size)	判断文件大小是否大于 0，如果是，返回值为 0
-c 文件名　(char)	判断文件是不是字符设备文件，如果是，返回值为 0
-e 文件名　(exist)	判断文件是否存在，如果是，返回值为 0

应用如例 5.8 所示。

```
[root@localhost ~]#[ -b jxnu ]          #判断文件是不是块设备文件
[root@localhost ~]#echo $?
1
[root@localhost ~]#[ -d jxnu ]          #判断文件是不是目录
[root@localhost ~]#echo $?
0
[root@localhost ~]#[ -f jxnu ]          #判断文件是不是普通文件
[root@localhost ~]#echo $?
1
[root@localhost ~]#[ -r jxnu ]          #判断文件是否可读
[root@localhost ~]#echo $?
0
[root@localhost ~]#[ -w jxnu ]          #判断文件是否可写
[root@localhost ~]#echo $?
0
[root@localhost ~]#[ -x jxnu ]          #判断文件是否可执行
[root@localhost ~]#echo $?
0
[root@localhost ~]#[ -s jxnu ]          #判断文件大小是否大于 0
[root@localhost ~]#echo $?
0
[root@localhost ~]#[ -c jxnu ]          #判断文件是不是字符设备文件
[root@localhost ~]#echo $?
1
[root@localhost ~]#[ -e jxnu ]          #判断文件是否存在
[root@localhost ~]#echo $?
0
```

例 5.8　文件测试运算符

5.4 Shell 变量

1. 变量的类型及定义

在 Linux 中，一般存在两种类型的变量：局部变量（local variable）和环境变量（environment variable）。局部变量在脚本或命令中定义，仅在当前 Shell 实例中有效，其他 Shell 启动的程序不能访问局部变量。环境变量存在于所有的程序中，包括 Shell 启动的程序，这些程序都能访问环境变量，有些程序需要环境变量来保证其正常运行。

在 Linux 下定义一个变量的方法是：变量名＝变量值。要特别注意的是，“＝”左右两边不能有空格。此外，通常会用 export 关键词定义环境变量，用 env 命令查看本地定义的所有环境变量。

变量名的命名须遵循如下规则。

（1）命名只能使用英文字母、数字和下画线，首个字符只能以英文字母开头。

（2）变量名中间不能有空格，可以使用下画线。

（3）不能使用标点符号。

（4）不能使用 bash 里的关键字。

应用如例 5.9 所示。

```
[root@localhost ~]#my_school=jxnu
[root@localhost ~]#echo my_school
my_school
```

例 5.9　查看变量类型

说明：这里使用 echo 命令查看变量 my_school 的值。

2. 变量的引用

使用一个定义过的变量，只要在变量名前面加符号“$”即可，也可以在变量名外面加上花括号，加花括号是为帮助解析器识别变量的边界。建议给所有变量加上花括号，这是一个好的编程习惯。

应用如例 5.10 所示。

```
[root@localhost ~]#my_school=jxnu
[root@localhost ~]#echo $my_school
jxnu
[root@localhost ~]#echo ${my_school}
jxnu
```

例 5.10　变量引用

说明：在使用 my_school 的值时，第一次没有加花括号，第二次加了花括号，解析器都能识别出变量的边界，但当有些变量较为复杂时，加上花括号可以让系统更快地识别。

3. 变量的重定义

重定义变量就是将已经被定义过的变量再次定义，重定义后的变量的值与之前被定义的值没有关系。在 Linux 中，已定义过的变量可以被重新定义。

应用如例 5.11 所示。

```
[root@localhost ~]#my_school=jxnu
[root@localhost ~]#echo $my_school
jxnu
[root@localhost ~]#my_school=JiangXiNormalUniversity
[root@localhost ~]#echo $my_school
JiangXiNormalUniversity
```

例 5.11　变量重定义

说明：一开始，将 my_school 的值定义为 jxnu，然后，再将 my_school 的值重定义为 JiangXiNormalUniversity，使用 echo 命令查看 my_school 的值，可以知道 my_school 已被重新定义了，并且重定义后的变量的值与之前被定义的值没有关系。

4. 只读变量

在 Linux 中，如果一个变量被 readonly 命令定义为只读变量，那么这个变量的值不能被修改。此外，可以使用 readonly -p 命令查看本地定义的所有只读变量。readonly 命令语句如下。

```
readonly 变量名
readonly -p
```

应用如例 5.12 所示。

```
[root@localhost /]#my_school=jxnu
[root@localhost /]#echo $my_school
jxnu
[root@localhost /]#readonly my_school
[root@localhost /]#my_school=JiangXiNormalUniversity
bash: my_school: readonly variable
[root@localhost /]#readonly -p
declare -r my_school="JiangXiNormalUniversity"
```

例 5.12　只读变量

说明：一开始，将 my_school 的值定义为 jxnu，再使用 readonly 命令定义 my_school，使得 my_school 变为只读变量，发现后面不能将 my_school 的值定义为 JiangXiNormalUniversity，这是因为只读变量的值不能被修改，因此，在写代码的过程中可以将重要的变量设置为只读变量，防止他人修改变量的值。

5. 变量的删除

使用 unset 命令可以删除变量，变量被删除后不能再使用，但是 unset 命令不能用来删除只读变量。unset 命令语句如下。

```
unset 变量名
```