尝试编写 Shell 脚本

洪艺中
数学与应用数学 3190105490

2022 年 6 月 28 日

在阅读了 30–60 页之后, 我发现关于 `$` 符号相关的引用格式是最为复杂的, 特别是其后面加括号的时候, 功能各不相同. 但是似乎这三十页中没有相关的 Try It Out, 有点可惜. 其他的 Try It Out 的内容都比较好理解, 于是我就选了一个看起来比较好玩的 48 页的 “Returning a Value”

1 代码和测试结果

文件 `original.sh` 即为我对书上例子的复现, 具体内容如下:

```
#!/bin/bash

yes_or_no() {
  echo "Is your name $* ?"
  while :
  do
    echo -n "Enter yes or no: "
    read x
    case "$x" in
      [yY] | [yY][eE][sS] ) return 0;;
      [nN][oO] ) return 1;;
      * ) echo "Answer yes or no";;
    esac
  done
}
```

```
echo "Original parameters are $*"

if yes_or_no "$1"
then
  echo "Hi $1, a good name"
else
  echo "Never mind"
fi

exit 0
```

调用时, 最好额外给出至少一个参数. 如可以输入 `bash original.sh a b` 来启动. 此外, 后面会有一步询问用户, 需要用户输入 `yes` 或者 `no`. 在前面的调用和yes的回答下, 输出为

```
$ bash original.sh a b
Original parameters are a b
Is your name a ?
Enter yes or no: yes
Hi a, a good name
$
```

接下来具体说明脚本的执行过程.

1. 首先会输出 `Original parameters are` 以及用户输入的全部参数, 当用户没有输入任何参数时, 后面也是空字符串.
2. 之后, 系统会输出 `Is your name` 加上输入的第一个参数 `$1`(当用户没有输入任何参数时, 这里也是空字符串), 也就是询问用户其输入的第一个参数「是不是用户的名字」, 并根据用户的回答给出不同的回答. 如果用户回答的不是 `y`, `yes`, `no` 的其中之一 (不区分大小写), 就会输出 `Answer yes or no` 并再次等待输入, 直到输入属于上面的三类.

2 学习心得

这一份脚本中包含了多个值得注意的细节, 下面逐一说明.

2.1 真值的写法

这个脚本中有一个 `while` 死循环, 书上原版用的是 `while true`, 但是后面它也讲到 `:` 是 “built-in” 的, 所以更为高效, 不过这种写法要牺牲可读性. 因此如果对速度要求很高, 应该选用 `:`.

2.2 函数

`Shell` 脚本中的函数定义格式和 `c++` 类似. 但是使用上有一些区别: `Shell` 中函数类似于定义了一个子可执行文件, 其调用方式和其他程序, 脚本以及命令类似, 直接按照 *函数名 函数参数列表* 的形式调用, 而非将函数的参数写在括号中. 同时, `Shell` 脚本的函数定义页不需要在括号中写明参数 (至少到书上 60 页是这样的).

因此, 取得函数参数的方法和取得程序调用输入的参数也是一样的, `$*` 代表了全部参数列表, `$1`, `$2` 等则对应第 1, 2 个参数.

2.3 case 结构

`Shell` 脚本中的 `case` 格式和 `c++` 也类似, 只是具体语法不同. 在使用 `case` 的时候, 有一些注意事项和技巧:

1. `case` 的同一分支的条件可以写的较为复杂, 可以利用 `|` 分割不同的判定条件. (似乎实际上是用一个正则表达式来判断, 所以功能还可以更丰富?)
2. `case` 一旦遇到满足的条件, 就会进入执行, 不会判断之后的其他可能更精确的条件. 因此必须把 `*` 这样含义广泛的条件放在后面.

2.4 参数调用

一般来说, 如果 `var` 是一个变量, `$var` 和 `"$var"` 的效果是类似的, 都是具体调用其值而非直接获得 `$var` 这个字符串. 但是在上面的脚本中

```
case "$x" in
```

和

```
if yes_or_no "$1"
```

这样的调用全都加了引号. 这也是书上介绍的一个避免**空输入引发程序错误**的技巧. 如果不加引号, 而且用户输入了空字符串, 那么在具体把 `$var` 转化为其内容是也会得到空的结果, 在有些时候 (比如使用 `[]` 这个判断命令时) 就会出错.

我认为 `$` 引用的机制和 `c++` 中宏定义类似, 是字符串的直接替换. 因此在调用参数值时, 需要考虑字符替换之后是否会引发其他错误. 加上引号之后, 无论结果是否为空, 得到的都是一个带引号括住的字符串, 就不会出现类似问题. 所以加上引号也是一个比较安全的好习惯.