


原

Shell中的数组及其相关操作

2016年07月26日 19:04:54

杰瑞26

阅读数: 36613

 版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/Jerry_1126/article/details/52027539

Shell中数据类型不多，比如说字符串，数字类型，数组。数组是其中比较重要的一种，其重要应用场景，可以求数组长度，元素长度，遍历其元素，删除等操作，使用非常方便。

Shell中的数组不像JAVA/C，只能是一维数组，没有二维数组;数组元素大小无约束，也无需先定义数组的元素个数;但其索引则像JAVA/C/Python其常用的方式进行总结。

【数组声明】

```
declare -a array_name      # 声明数组，也可不声明
declare -a nums=(1 2 3 4)  # 声明数组，同时也可赋值
unset array_name           # 删除数组，撤销数组
unset nums[0]              # 删除数组中的某个元素
```

备注:

- 1) 不像JAVA/C等强编程语言，在赋值前必须声明；SHELL只是弱编程语言，可事先声明也可不声明；
- 2) 用unset来撤销数组，可用unset array_name[i]来删除里面的元素

【数组定义】

```
# 方式一:
array_name=(
value0
value1
value2
value3
)

# 方式二:
names=(Jerry Alice David Wendy)

# 方式三:
names[0]=Jerry
names[1]=Alice
names[2]=David
names[3]=Wendy

# 方式四:
names=([0]=Jerry [1]=Alice [2]=David [3]=Wendy)

# 方式五:
str="Jerry Alice David Wendy"
names=($str)
```

备注:

- 1) 数组中的元素，必须以"空格"来隔开，这是其基本要求；
- 2) 定义数组其索引，可以不按顺序来定义，比如说:names=([0]=Jerry [1]=Alice [2]=David [8]=Wendy);
- 3) 字符串是SHELL中最重要的数据类型，其也可通过(\$str)来转成数组，操作起来非常方便；

【数组长度】

```

### 求数组长度:
nums=(Jerry Alice David Wendy)
# 方式一: ${#数组名[0]}
${#names[0]} # 输出结果: 4, 4个元素
# 方式二: ${#数组名[*]}
${#nums[*]} # ${array[0]}等价于${array[*]}

### 求元素长度:
nums=(Jerry Alice David Wendy)
# 方式一: ${#数组名[index]}
${#nums[0]} # 求第一个元素的长度
# 方式二: expr length ${数组名[index]}
expr length ${names[0]} # expr length的函数来求元素长度
# 方式三: wc -L
echo ${names[0]} | wc -L # 使用-L参数来获取当前行的长度
# 方式四: wc -l结合echo -n
echo -n ${names[0]} | wc -l # -n参数, 去除"\n"换行符
# 方式五: expr ${names[0]} : ".*"
expr ${names[0]} : ".*" # .*代表任意字符, 用任意字符来匹配字符串
# 方式六: awk的NF项
echo ${names[0]} | awk -F " " '{print NF}' # 先分隔, 再用NF域来求元素长度
# 方式七: awk的length
echo ${names[0]} | awk '{print length($0)}' # 使用awk中的length()函数

```

备注:

- 1) 使用\${array_name[@]} 或者 \${array_name[*]} 都可以全部显示数组中的元素
- 2) 同样道理\${#array_name[@]} 或者 \${#array_name[*]}都可以用来求数组的长度
- 3) 求数组中元素的长度方法有很多, 相当于求字符串的长度

【数组索引】

```

1 [root@localhost ~]# s="A,B,C,D"
2 [root@localhost ~]# a=(`echo $s | tr ' ,' ' ') # 将字符串转变成数组
3 [root@localhost ~]# echo ${!a[@]} # 求数组中的索引
4 0 1 2 3

```

【元素删除】

```

1 [root@localhost ~]# a=(A B C D)
2 [root@localhost ~]# unset a[2] # 删除索引为2的元素
3 [root@localhost ~]# echo ${a[@]} # 显示删除后的元素
4 A B D

```

【数组遍历】

```

#!/bin/bash

# 方式一: 按索引来遍历
names=(Jerry Alice David Wendy)
for ((i=0;i<${#names[*]};i++))
do
    echo ${names[$i]}
done

# 方式二: 不按索引来遍历
names=(Jerry Alice David Wendy)
index=0
for i in ${names[@]}
do
    echo "第${index}个元素的值为: ==> ${i}"
    let index++
done

```

脚本输出:

Python实战训练

转型AI人工智能指南

数据库沙龙

21 天入门机器学习

盒马 招聘

生鲜配送

登录

注册

×

```
root@localhost:~/training# vim test.sh
root@localhost:~/training# ./test.sh
Jerry
Alice
David
Wendy
第0个元素的值为: ==> Jerry
第1个元素的值为: ==> Alice
第2个元素的值为: ==> David
第3个元素的值为: ==> Wendy
```

备注:

- 1) 可以使用标准的for循环, 这种类C语言的方式来遍历数组中的元素
- 2) for 元素 in 元素集(数组) 这种类Python的方式来遍历数组
- 3) 从代码可读性与执行速度来看, 推荐使用第二种方式

【数组赋值】

```
nums=(1 2 3 4)      # 定义一个数组
nums[3]=44           # 给第三个元素重新赋值
echo ${nums[@]}      # 结果变成了1 2 3 44
```

备注:

- 1) 第一种是给已经存在的元素项重新赋值
- 2) 当然也可以给不存在的索引添加赋值, 可以看下面的示例

【数组添加】

```
### 数组添加
# 方式一: 直接赋值给不在的索引一个值
nums=(1 2 3 4)      # 定义一个数组
nums[4]=5           # 给第四个新元素赋值
echo ${nums[@]}      # 结果变成了1 2 3 4 5

# 方式二: 直接使用 新数组=(旧数组 新元素) 的方式来添加元素
old=(1 2 3 4)
new=(${old[*]} 5)
echo ${new[@]}
```

【数组切片】

数组切片

```
### 数组切片
array=(zero one two three four)
$array              # 默认取第一个元素, 输出:zero
${array[0]}         # 取索引为零对应的元素, 输出:zero
${array[@]}         # 取数组中全部元素
${array[@]:1}       # 从索引为1到后面所有的值, 输出:one two three four
${array[@]:0:3}     # 从索引为0开始取起, 共取三位, 输出:zero one two
${array[@]::4}      # 从索引为0开始取起, 共取四位, 输出:zero one two three
${array[@]:(-2):2}  # 从倒数第二个元素开始起, 取两位, 输出:three four
new_array=(${array[@]:1:4}) # 得到新的切片数组
```

元素切片

```
root@localhost:~# names=(Jerry Alice David Wendy)
root@localhost:~# echo ${names[0]:0}      # 取索引为开始取所有
Jerry
root@localhost:~# names=(Jerry Alice David Wendy)
root@localhost:~# echo ${names[0]:0}      # 取索引为0开始取所有
Jerry
root@localhost:~# echo ${names[0]:1}      # 取索引为1开始取所有
erry
root@localhost:~# echo ${names[0]:2:2}    # 取索引为2开始取两位
rr
root@localhost:~# echo ${names[0]:6}      # 超出元素长度显示空行
```

备注:

- 1) 通用的格式\${array[@]:起始位置:长度}, 中间以":"隔开, 如果第二项省略的话, 就取后面所有的项
- 2) 切片后返回的是字符串, 可以通过 新数组=(\${旧数组[@]:索引:长度})来索引, 参见上面最后一个例子
- 3) 区别于Python之一:起始位置可以为负数, 但必须以放在()中, 长度不能为负数
- 4) 区别于Python之二:第二项在Python里面是结束索引, 在Shell则代表所取元素的长度
- 5) 区别于Python之三:Python可以通过 list[-1:-4:-2]来反向取数, 在Shell则实现不了

【数组替换】

\${array[@]/x/y} 最小匹配替换, 每个元素只替换一次

\${array[@]//x/y} 最大匹配替换, 每个元素可替换多次

\${array[@]/x/} 最小匹配删除, 只删除一个符合规定的元素

\${array[@]//x/} 最大匹配删除, 可删除多个符合规定的元素

```
root@localhost:~# array=(one two three four)
root@localhost:~# echo ${array[@]/e/E}    # 每个元素只替换一次,thrEe
onE two thrEe four
root@localhost:~# echo ${array[@]//e/E}   # 每个元素替换多次,thrEE
onE two thrEE four
root@localhost:~# echo ${array[@]/e/}     # 最小匹配删除,on thre
on two thre four
root@localhost:~# echo ${array[@]//e/}    # 最大匹配删除,on thr
on two thr four
```

\${array[@]/#x/y} 从左往右匹配替换, 只替换每个元素最左边的字符

\${array[@]/%x/y} 从右往左匹配替换, 只替换每个元素最右边的字符

```
root@localhost:~# array=(one two three four)
root@localhost:~# echo ${array[@]/#o/O}   # 只替换每个元素左边的o,One
One two three four
root@localhost:~# echo ${array[@]/%o/O}   # 只替换每个元素右边的o,two
one twoO three four
```

【数组删除】

每个元素,从左向右进行最短匹配

每个元素,从左向右进行最长匹配

% 每个元素,从右向左进行最短匹配

%% 每个元素,从右向左进行最长匹配

```
root@localhost:~# list=(book food)
root@localhost:~# echo ${list[@]#b*o}      # bo匹配到被删除
ok food
root@localhost:~# echo ${list[@]##b*o}      # boo匹配到被删除
k food
root@localhost:~# echo ${list[@]%o*d}      # od匹配到被删除
book fo
root@localhost:~# echo ${list[@]%%o*d}     # ood匹配到被删除
book f
```

【数组应用】

示例一: 将ifconfig命令取到的本地IP: 127.0.0.1逐行显示出来

```
#!/bin/bash
# 将内容为: inet addr:127.0.0.1 Mask:255.0.0.0中的127.0.0.1逐行打印出来
#
STR_RAW=`ifconfig | grep -A 1 '^lo' | grep 'inet addr:'`
STR_NEW=`echo $STR_RAW | awk '{print $2}' | awk -F':' '{print $2}'`

ARRAY=(`echo $STR_NEW | tr '.' ' '`)

for element in ${ARRAY[@]}
do
    echo ${element}
done
```

脚本输出:

```
root@localhost:~/training# ./test.sh
127
0
0
1
```

示例二: 模拟堆栈的push,pop,shift,unshift操作

```
#!/bin/bash
# 用数组来模拟堆栈操作(pop push shift unshift等)

array=(1 2 3 4)
echo "原始数组==> ${array[@]}"

# 模拟堆栈push操作
array=(${array[@]} 5)
echo "模拟堆栈push操作后输出==> ${array[@]}"

# 模拟堆栈pop操作
array=(${array[@]:0:${#array[@]}-1})
echo "模拟堆栈pop操作后输出==> ${array[@]}"

# 模拟堆栈shift操作
array=(${array[@]:1})
echo "模拟堆栈shift操作后输出==> ${array[@]}"

# 模拟堆栈unshift操作
array=(0 ${array[@]})
echo "模拟堆栈unshift操作后输出==> ${array[@]}"
```

脚本输出:

```
root@localhost:~/training# ./test.sh
原始数组==> 1 2 3 4
模拟堆栈push操作后输出==> 1 2 3 4 5
模拟堆栈pop操作后输出==> 1 2 3 4
模拟堆栈shift操作后输出==> 2 3 4
模拟堆栈unshift操作后输出==> 0 2 3 4
```

示例三: 在1-10间, 随机生成10个不重复的数, 将其放置于数组中

```
#!/bin/bash

declare -a array
count=0
while [ $count -lt 10 ]; do
    random=$((RANDOM%10+1))
    if [ -z $(grep -F $random -F array) ]; then
        array+=($random)
        count=$((count+1))
    fi
done
```

脚本输出:

```
root@localhost:~/training# ./test.sh
数组中随机生成10个数为:-> 8 5 3 9 7 4 2 6 10 1
```

备注:

- 1) 生成[1,10]范围内不重复的随机整数, 并保存到数组array中
- 2) seq 1 10 用于生成1~10的整数序列(包含边界值1和10)
- 3) awk中的rand()函数用于随机产生一个0到1之间的小数值(保留小数点后6位)
- 4) rand()只生成一次随机数, 要使用srand()函数使随机数滚动生成
- 5) 括号里留空即默认采用当前时间作为随机计数器的种子, 这样以秒为间隔, 随机数就能滚动随机生成了
- 6) 由于以秒为间隔, 所以如果快速连续运行两次脚本(1s内), 你会发现生成的随机数还是一样的

示例四: 将字符串处理后转为为数组, 再对其打印输出

```
#!/bin/bash
# 将字符串'2016-12-31_08:10:30_ABCD.log'中的数字提取出来

str=`echo 2016-12-31_08:10:30_ABCD.log | awk -F. '{print $1}' \
    | sed -e 's/[-_:/ /g' | tr -d [a-zA-Z]`
arr=(${str})
for element in ${arr[@]}
do
    echo $element
done
```

脚本输出:

```
root@localhost:~/training# ./test.sh
2016
12
31
08
10
30
```

示例五: 用read -a参数, 从标准输入中读取数组, 再做操作

```
#!/bin/bash

echo -n "Please Enter the names: "

read -a names

i=0
while [ $i -lt ${#names[@]} ]
do
    echo ${names[$i]}
    let i++
done
```

脚本输出:

```
root@localhost:~/training# ./test.sh
Please Enter the names: Jerry Alice David Wendy
Jerry
Alice
David
Wendy
```

示例六: 判断某个变量, 是否在数组中, 在输出YES, 否输出NO

```
#!/bin/bash

names=(Jerry Alice David Wendy)
var=Tom
[[ ${names[@]/$var/} != ${names[@]} ]] && echo 'Yes' || echo 'No'
```

```
root@localhost:~/training# ./test.sh
No
```

示例七: 对数组中的元素进行排序

```
root@localhost:~# old=(7 1 4 101)
root@localhost:~# new=(`echo ${old[@]} | tr ' ' '\n' | sort -n`)
root@localhost:~# echo ${new[@]}
1 4 7 101
```

示例八: 将/etc/passwd文件中以:分隔的第一列,即用户名放置于一个数组中

```
#!/bin/bash

i=0
while read line
do
    arr[${i}]=`echo ${line} | awk -F":" '{print $1}'`
    (( ++i ))
done < /etc/passwd
echo "${arr[@]}"
```

示例九: 将1-8, 每个数自乘后输出

```
#!/bin/bash

array=({1..8})
for ((i=0;i<8;i++))
do
    declare -i result=${array[$i]}*${array[$i]}
    echo $result
done
```

脚本输出:

```
root@localhost:~/training# ./test.sh
1
4
9
16
25
36
49
64
```

示例十: 借助数组来设置SHELLS的环境变量

```
root@localhost:~# set | grep "SHELLS" # 当前未设置
root@localhost:~# cat /etc/shells # 所有可用的SHELLS
# /etc/shells: valid login shells
/bin/sh
/bin/dash
/bin/bash
/bin/rbash
/bin/zsh
/usr/bin/zsh
root@localhost:~# more +2 /etc/shells|tr '\n' ' ' > file # 除首行外, 重定向到一文件中
root@localhost:~# read -a SHELLS < file # 将刚才的文件赋值给一数组中
root@localhost:~# set | grep "SHELLS" # 此环境变量成为公共环境变量
SHELLS=( [0]="/bin/sh" [1]="/bin/dash" [2]="/bin/bash" [3]="/bin/rbash" [4]="/bin/zsh" [5]="/usr/bin/zsh")
```

示例十一: 设置IFS, 读取文件内容示例



6



4



```
#!/bin/bash

# 设置IFS, 将分隔符设置为换行符
OLDIFS=$IFS
IFS=$'\n'

# 读取文件中的内容到数组中
array=($(cat hosts))

# 恢复之前的设置
IFS=$OLDIFS
LEN=${#array[@]}

# 循环显示文件的内容
i=0
while [[ $i -lt $LEN ]]
do
    echo ${array[$i]}
    let i++
done
```



6



4



示例十二：利用eval，模拟实现数组的功能

```
#!/bin/bash

user1="jerry 001 21"
user2="alice 002 18"
user3="wendy 003 35"
for user in user1 user2 user3
do
    eval array=\${$user} # 注意eval的用法
    for item in ${array[@]}
    do
        echo $item
    done
done
```

脚本输出:

```
root@localhost:~/training# ./test.sh
jerry
001
21
alice
002
18
wendy
003
35
```

示例十三：利用数组来实现冒泡排序

思路:会重复地走访过要排序的数组，一次比较两个元素，如果他们的顺序错误就把他们交换过来。走访数列的工作是重复地进行直到没有再需要交换，也就数列已经排序完成。越大的元素会经由交换慢慢“浮”到数列的顶端

```
#!/bin/bash

echo "Please input a number list: "

read -a arr

for ((i=0;i<${#arr[@]};i++))
do
    for ((j=${#arr[@]}-1;j>i;j--))
    do
        if [[ ${arr[j]} -lt ${arr[j-1]} ]]
        then
            temp=${arr[j]} # 设置临时变量交换
            arr[j]=${arr[j-1]}
            arr[j-1]=$temp
        fi
    done
done
echo "排序后的输出为--> "
echo ${arr[@]}
```

[Python实战训练](#)[转型AI人工智能指南](#)[数据库沙龙](#)[21 天入门机器学习](#)[盒马 招聘](#)[生鲜配送](#)[登录](#)[注册](#)