11 数组基础

数组基础

一、数组的概念

1.1 数组的实质

数组是一种数据结构，是**相同数据类型**的元素按一定**顺序排列**的**元素集合**。

数组实际上就是**一连串类型相同的变量**，这些变量用**数组名命名**，并用**索引**互相**区分**。

Shell数组对**元素个数没有限制**，但**只支持一维数组**，这一点和很多语言不同。

数组的典型应用场景是**一次性要记录很多类型相同的数据**。比如，用数组记录班级中所有人的数学成绩。

1.2 数组的类别

数组分为**普通数组**和**关联数组**。

使用数组时，可以通过**索引**来**访问数组元素**，如数组元素的赋值和取值。

索引有时也称为**数组下标**，所以数组的元素也称为下标变量。

普通数组中的索引（下标）都是**整数**；关联数组的数组索引可以用任意的**文本**。

普通数组类似于Python中的**列表**；关联数组由**键值对**组成，类似于Python中的**字典**。

1.3 数组的声明

关联数组需要**先声明再使用**，语法格式为：`declare -A array`。

通常情况下Shell解释器**隐式声明普通数组**，**用户无须操作**。若用户需显式声明普通数组，语法格式为：`declare -a array`。

二、数组的定义

Shell有多种数组的定义方法：直接定义数组、键值对定义数组、间接定义数组和从文件中读入定义数组。

2.1 直接定义普通数组

普通数组中数组元素的索引(下标)**从0开始编号**，获取数组中的元素要利用索引(下标)。索引(下标)可以是算术表达式，其结果必须是一个整数。

在 Shell 中，**用括号 `( )` 来表示数组**，数组元素之间用**空格分隔**。由此，定义数组的一般形式为：

```shell
array_name=(ele1  ele2  ele3 ... elen)
```

注意，**赋值号 `=` 两边不能有空格**，必须紧挨着数组名和数组元素。

案例：直接定义普通数组

```shell
#错误示例：=两边有空格
[root@Shell ~]# books = (Linux Shell awk openstack docker)
-bash: syntax error near unexpected token (
#正确示例
[root@Shell ~]# books=(Linux Shell awk openstack docker)
[root@Shell ~]# echo $books
Linux
# 错误示例
[root@Shell ~]# echo $books[3]
Linux[3]
# 使用下标访问元素
[root@Shell ~]# echo ${books[3]}
openstack
```

2.2 键值对定义数组

除了直接定义数组，也可以采用采用键值对的形式赋值。**普通数组和关联数组均支持**。

方括号里对应的值为数组**索引**，**等号后面**的内容为索引对应的数组变量的**值**。

键值对形式定义数组格式如下。

```shell
array_name=([1]=value1 [2]=value2  [3]=value3……)
数组名= ([索引1]=变量值1 [索引2]=变量值2 [索引3]=变量值3...)
```

案例：键值对形式定义普通数组

```shell
[root@Shell ~]# books2=([0]=Linux [1]=Shell [2]=awk [3]=openstack [4]=docke
r)
[root@Shell ~]# echo ${books[3]}
openstack
```

案例：键值对形式定义关联数组

```shell
[root@Shell ~]# declare -A info1
[root@Shell ~]# info1=([name]=tianyun [sex]=male [age]=36 [height]=170 [ski
ll]=cloud)
[root@Shell ~]# echo ${info1[age]}
36
```

2.3 间接定义数组

间接定义数组通过**定义每个数组元素**的方法来定义数组，语法格式如下。

```shell
array_name[0]=value1;array_name[1]=value2;array_name[2]=value3
# 数组名[索引]=变量值
```

这种方法要求一次赋一个值，比较繁琐。

案例：间接定义普通数组

```shell
[root@Shell ~]# array[0]=pear
[root@Shell ~]# array[1]=apple
[root@Shell ~]# array[2]=orange
[root@Shell ~]# array[3]=peach
[root@Shell ~]# echo ${array[0]}
pear
```

案例：间接定义关联数组

```shell
[root@Shell ~]# declare -A weekdays
[root@Shell ~]# weekdays[a]=monday
[root@Shell ~]# weekdays[b]=tuesday
[root@Shell ~]# echo ${weekdays[a]}
monday
```

2.4 从文件中读入定义数组

从文件中读入定义数组是指使用**命令的输出结果**作为**数组的元素**。

分隔符(由IFS决定，默认为空格、TAB或换行)间隔的每一部分输出为数组的一个元素。

其语法格式为：`数组名=($(命令))` 或：`数组名=(`命令`)` 。

案例：通过文件生成数组

```shell
[root@Shell ~]# cat user1.txt
abcd    123
qwer    123
# 注意！此处设置分隔符为换行
[root@Shell ~]# IFS=$'\n'
[root@Shell ~]# users=(`cat user1.txt`)
[root@Shell ~]# echo ${users[1]}
qwer    123
[root@Shell ~]# users=(`cat /etc/passwd`)
[root@Shell ~]# echo ${users[1]}
bin:x:1:1:bin:/bin:/sbin/nologin
```

三、操作数组

3.1 数组索引

常见的数组索引表达式如下。

语法	描述
`${!array[*]}`	访问数组所有索引
`${!array[@]}`	访问数组所有索引
`${array[*]}`	访问数组所有元素

`${array[@]}`	访问数组所有元素
`${array[0]}`	访问数组中的第1个元素
`${array}`	访问数组中的第1个元素
`${array[n]}`	访问数组中的第n+1个元素
`${#array[@]}`	统计数组元素个数
`${#array}`	统计数组索引为0的元素的字符个数
`${#array[n]}`	统计数组索引为n的元素的字符个数

Shell 复制代码

```shell
[root@Shell ~]# books=(Linux Shell awk openstack docker)
#访问数组所有索引
[root@Shell ~]# echo ${!books[*]}
0 1 2 3 4
[root@Shell ~]# echo ${!books[@]}
0 1 2 3 4
#访问数组所有元素
[root@Shell ~]# echo ${books[*]}
Linux Shell awk openstack docker
[root@Shell ~]# echo ${books[@]}
Linux Shell awk openstack docker
#访问数组第一个元素
[root@Shell ~]# echo ${books}
Linux
# 统计数组元素个数
[root@Shell ~]# echo ${#books[@]}
5
#统计数组第一个元素的字符个数
[root@Shell ~]# echo ${#books}
5
#统计数组第三个元素的字符个数
[root@Shell ~]# echo ${#books[2]}
3
```

3.2 数组的赋值

Shell通过 `数组名[索引]` 对数组进行引用赋值，**如果下标不存在，则自动添加一个新的数组元素；如果下标存在，则覆盖原来的值**。

数组的赋值语法格式为：

```shell
$array_name[index1]=value1
$array_name[index2]=value2
数组名[索引]=变量值
```

案例：数组赋值

```shell
[root@Shell ~]# array=(tom lucy alice)
[root@Shell ~]# echo ${array[*]}
tom lucy alice
[root@Shell ~]# echo ${array[2]}
alice
#数组赋值
[root@Shell ~]# array[2]=lily
[root@Shell ~]# echo ${array[2]}
lily
```

`array` 数组赋值为tom、lucy和alice。用 `echo $(array[*])` 打印出数组的元素，用 `${array[2]}` 打印出第二个元素，用 `array[2]=lily` 修改数组元素为lily，打印出第二个元素就是已经修改的元素。

3.3 数组的删除

Shell通过 `unset 数组[索引]` 删除相应数组元素，**如果不带下标，则表示删除整个数组的所有元素。**

案例：删除数组元素

```shell
[root@Shell ~]# array=(tom lucy alice)
[root@Shell ~]# echo ${array[*]}
tom lucy alice
#删除下标为1的数组元素
[root@Shell ~]# unset array[1]
[root@Shell ~]# echo ${array[*]}
tom alice
#lucy已被删除
#删除整个数组
[root@Shell ~]# unset array
[root@Shell ~]# echo ${array[*]}
#输出为空
[root@Shell ~]#
```

3.4 数组的截取

Shell通过 `${数组名[@或*]:起始位置:长度}` 切片原先数组，**返回值**的是**字符串**，中间用空格分开，如果加上 `()` ，将得到切片数组。

案例：数组截取

```shell
[root@Shell ~]# array=(1 2 3 4 5 6 7 8)
[root@Shell ~]# echo ${array[@]:0:3}
1 2 3
[root@Shell ~]# echo ${array[@]:1:4}
2 3 4 5
# 切片数组
[root@Shell ~]# c=(${array[@]:1:4})
[root@Shell ~]# echo ${c[*]}
2 3 4 5
[root@Shell ~]# echo ${#c[@]}
4
```

3.5 数组的替换

Shell通过 `${数组名[@或*]/查找字符/替换字符}` 替换数组，**返回值**的是**字符串**。

案例：数组替换

```shell
[root@Shell ~]# array=(1 2 3 4 5 6 7 8)
#将3替换为100
[root@Shell ~]# echo ${array[@]/3/100}
1 2 100 4 5 6 7 8
[root@Shell ~]# echo ${array[@]}
1 2 3 4 5 6 7 8
[root@Shell ~]# a=(${array[@]/3/100})
[root@Shell ~]# echo ${a[@]}
1 2 100 4 5 6 7 8
```

3.6 数组的遍历

遍历数组的方法有多种，利用for循环直接遍历元素或通过索引遍历元素较为常用。

案例：直接遍历元素

```shell
[root@Shell ~]# vi for_array.sh
#!/bin/bash
#for循环默认以 tab、空格和回车为换行符，所以要提前定义变量以行作为分隔符
OLD_IFS=$IFS
IFS=$'\n'
#读取hosts文件，并赋值数组
hosts=(`cat /etc/hosts`)
#遍历数组元素
for i in ${hosts[@]}
do
    echo "$i"
done
[root@Shell ~]# . for_array.sh
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
```

⚠️ **遍历数组所有元素一定要使用 `${hosts[@]}` 或 `${hosts[*]}` 表达式。**

案例：通过索引遍历数组

```shell
[root@Shell ~]# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
[root@Shell ~]# vi while_array.sh
#!/bin/bash
#初始化对象，避免多次运行影响结果
i=0
hosts=()
#读取hosts文件，并赋值数组
while read line
do
    hosts[++i]=$line
done </etc/hosts
echo "hosts first: ${hosts[1]}"
#打印一个空行
echo
#获得数组的索引，并遍历元素
for i in ${!hosts[@]}
do
    echo "$i: ${hosts[i]}"
done
[root@Shell ~]# . while_array.sh
hosts first: 127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4

1: 127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
2: ::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
```

本案例以 `hosts` 文件的每一行作为数组的一个元素来做赋值，并对该数组进行遍历。 `while` 读入 `/etc/hosts` 文件中的每一行并把它显示出来， `hosts[++i]=$line` 这个表达式完成数组的赋值操作， `${!hosts[@]}` 这个表达式获得数组的索引， `${hosts[i]}` 结合 `while` 循环这个表达式完成了数组的遍历。

小结

- 数组的概念：普通数组、关联数组、索引、定义

- 数组的定义：直接、键值对、间接、文件
- 操作数组：索引、复制、删除、截取、替换、

课程目标

- 知识目标：熟练掌握数组的基本概念。
- 技能目标：能够根据实际需求定义数组、操作数组、遍历数组。

课外拓展

- 进一步了解数组的应用场景。

参考资料

- `man bash` `/arrays`
- 《Linux Shell核心编程指南》，丁明一，电子工业出版社